ECG CLASSIFICATION WITH AN ADAPTIVE NEURO-FUZZY INFERENCE SYSTEM

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science and
Master of Science in Electrical Engineering

by
Brad Thomas Funsten
June 2015

COMMITTEE MEMBERSHIP

TITLE: ECG Classification with an Adaptive Neuro-Fuzzy Inference System

AUTHOR: Brad Thomas Funsten

DATE SUBMITTED: June 2015

COMMITTEE CHAIR: Xiao-Hua (Helen) Yu, Ph.D.
Professor of Electrical Engineering

COMMITTEE MEMBER: Jane Zhang, Ph.D.
Professor of Electrical Engineering

COMMITTEE MEMBER: John Saghri, Ph.D.
Professor of Electrical Engineering

ACKNOWLEDGMENTS

I would like to thank my friends and family for their enormous support in completing this thesis. I would like to thank Dr. Yu for her advice and counseling in accomplishing a project such as this. To all the engineers at Advanced Telesensors, Inc., thank you so much for your insight that paved the way for the thesis. I want to thank my doctor, Dr. Gazarian, for giving me a free ECG test.

I want to emphasize that completing this thesis would not be possible without the support of friends and family. Whether it was school, church, or home, the power of encouragement and fun times allowed me to persevere. Proofreading from family and friends was much appreciated. Thank you all!

ABSTRACT

ECG Classification with an Adaptive Neuro-Fuzzy Inference System

Brad Thomas Funsten

Heart signals allow for a comprehensive analysis of the heart. Electrocardiography (ECG or EKG) uses electrodes to measure the electrical activity of the heart. Extracting ECG signals is a non-invasive process that opens the door to new possibilities for the application of advanced signal processing and data analysis techniques in the diagnosis of heart diseases. With the help of today's large database of ECG signals, a computationally intelligent system can learn and take the place of a cardiologist. Detection of various abnormalities in the patient's heart to identify various heart diseases can be made through an Adaptive Neuro-Fuzzy Inference System (ANFIS) preprocessed by subtractive clustering. Six types of heartbeats are classified: normal sinus rhythm, premature ventricular contraction (PVC), atrial premature contraction (APC), left bundle branch block (LBBB), right bundle branch block (RBBB), and paced beats. The goal is to detect important characteristics of an ECG signal to determine if the patient's heartbeat is normal or irregular. The results from three trials indicate an average accuracy of 98.10%, average sensitivity of 94.99%, and average specificity of 98.87%. These results are comparable to two artificial neural network (ANN) algorithms: gradient descent and Levenberg Marquardt, as well as the ANFIS preprocessed by grid partitioning.

Keywords: ECG, Fuzzy Logic, Adaptive Neuro Fuzzy Inference System, ANFIS, Computational Intelligence, Neural Network, ANN, Signal Processing

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES xi

CHAPTER

1: INTRODUCTION 1

1.1: Project Goal 1

1.2: Overview 1

1.3: Software Environment 1

1.4: Research Importance 2

1.5: Past Research 4

1.6: Brief Approach 4

2: BACKGROUND 6

2.1: Electrocardiography 6

2.2: Human Heart 9

2.3: Heart Abnormalities 11

2.4: Fuzzy Set Theory 16

2.5: ANFIS Algorithm 20

2.6: Hybrid Learning Algorithm 25

2.7: ANFIS Overview 31

2.8: Subtractive Clustering as a Preprocessor to ANFIS Classification 32

3: LITERATURE REVIEW 37

3.1: Time Domain Based Classifier for ECG Signals 37

3.2: Radial-Basis-Function (RBF) Neural Network with the Genetic Algorithm for ECG Classification 38

3.3: Hyperellipsoidal Classifier for ECG Classification 40

3.4: Adaptive Parameter Estimation Using Sequential Bayesian Methods for ECG Classification....42
3.5: Fast Fourier Transform and Levenberg-Marquardt Classifier for ECG Signals....44
3.6: Support Vector Machine (SVM) and Wavelet Decomposition for ECG Classification....46
3.7: Past Research on ANFIS for ECG Classification....47
4: CLASSIFICATION APPROACH....50
4.1: ANFIS of ECG Signals....50
4.2: Extracting Input Features....55
4.3: Program Flow....63
4.4: Performance Evaluation Method....67
5: SIMULATION RESULTS....68
5.1: ANFIS on Several ECG Records under Subtractive Clustering....68
5.2: Justification of Chosen Input Records....85
5.3: Comparison to ANN....86
5.4: Comparison to ANFIS under Grid Partitioning....96
6: CONCLUSIONS AND FUTURE WORKS....104
REFERENCES....106
APPENDICES....109
APPENDIX A – List of Acronyms....109
APPENDIX B – MATLAB Code....110
Plot ECG Signal from MIT-BIH Arrhythmia Database with Annotations:....110
ECG Input Layer:....114
Input to ANFIS for One ECG Signal:....139
Input to ANFIS for Multiple ECG Signals:....140
ANFIS (grid partitioning or subtractive clustering) for ECG Signals and Evaluation of Test Data:....150

Comparison with ANN (Gradient Descent and Levenberg-Marquardt: 155

APPENDIX C – Project Analysis.....157

LIST OF TABLES

Table Page

1: Hybrid learning applied to ANFIS [15] 25

2: Input features of an ECG signal [5], [8], [24] 52

3: MIT-BIH records and corresponding heartbeat types: ‘N’ being normal, ‘V’ being PVC, ‘A’ being APC, ‘L’ being LBBB, ‘R’ being RBBB, and ‘P’ being paced beats 58

4: The seven features associated with a corresponding input number 68

5: Heartbeats (including each heartbeat type) chosen from records of the MIT-BIH database randomly 69

6: Trial 1 training and checking results for the six ANFIS’ 78

7: Trial 1 classification results for the six ANFIS’ 79

8: Analysis of a PVC and LBBB heartbeat that were classified incorrectly 80

9: Trial 2 training and checking results for the six ANFIS’ 82

10: Trial 2 classification results for the six ANFIS’ 82

11: Trial 3 training and checking results for the six ANFIS’ 84

12: Trial 3 classification results for the six ANFIS’ 84

13: Average accuracy, sensitivity, and specificity results for 3 trials for subtractive clustering preprocessed ANFIS 85

14: Trial training and checking results for the six ANFIS training and checking data for gradient descent 88

15: Trial 1 classification results for the six gradient descent neural networks (NN) 89

16: Trial 2 training and checking results for the six ANFIS training and checking data for gradient descent 89

17: Trial 2 classification results for the six gradient descent neural networks (NN) 90

18: Trial 3 training and checking results for the six ANFIS training and checking data for gradient descent 90

19: Trial 3 classification results for the six gradient descent neural networks (NN) 91

20: Average accuracy, sensitivity, and specificity results for 3 trials for gradient descent 91

21: Trial 1 training and checking results for the six ANFIS training and checking data for Levenberg-Marquardt 93

22: Trial 1 classification results for the six Levenberg-Marquardt neural networks (NN).....93

23: Trial 2 training and checking results for the six ANFIS training and checking data for Levenberg-Marquardt 94

24: Trial 2 classification results for the six Levenberg-Marquardt neural networks (NN).....94

25: Trial 3 training and checking results for the six ANFIS training and checking data for Levenberg-Marquardt 94

26: Trial 3 classification results for the six Levenberg-Marquardt neural networks (NN).....95

27: Average accuracy, sensitivity, and specificity results for 3 trials for Levenberg-Marquardt 95

28: Trial 1 training and checking results for the six grid partitioned ANFIS' 99

29: Trial 1 classification results for the six gradient partitioned ANFIS' 100

30: Trial 2 training and checking results for the six grid partitioned ANFIS' 101

31: Trial 2 classification results for the six grid partitioned ANFIS'.....101

32: Trial 3 training and checking results for the six grid partitioned ANFIS' 101

33: Trial 3 classification results for the six grid partitioned ANFIS'.....102

34: Average accuracy, sensitivity, and specificity results for 3 trials for grid partitioned ANFIS 102

35: Different membership function type average accuracy, sensitivity, and specificity results For 3 trials for grid partitioned ANFIS 103

36: Average accuracy, sensitivity, and specificity results for 3 trials for several algorithms 104

LIST OF FIGURES

Figure Page

1: ECG signal showing two PVCs detected and corresponding degree of fulfilment versus ANFIS decision rule [7] 3

2: ECG signal [23] 6

3: ECG device 7

4: ECG Paper 7

5: Ten electrodes (Twelve-leads) [9] 8

6: The human heart [27] 9

7: Illustrative ECG relation to the human heart [27] 10

8: Example of a PVC beat from MIT-BIH database, record 100, lead: Modified Limb Lead II (MLII) 11

9: Example of an APC beat from MIT-BIH database, record 100, lead: Modified Limb Lead II (MLII) 12

10: Example of a LBBB from MIT-BIH database, record 109, lead: Modified Limb Lead II (MLII) 13

11: Example of a RBBB from MIT-BIH database, record 118, lead: Modified Limb Lead II (MLII) 14

12: Example of a paced beat from MIT-BIH database, record 104, lead: Modified Limb Lead II (MLII) 15

13: Several membership functions μ to choose from for a fuzzy set Z 17

14: FIS diagram [12] 18

15: n-input first-order Sugeno Adaptive Neuro-Fuzzy Inference System (ANFIS) [12] 23

16: Two-input, two-rule, first order Sugeno ANFIS model [12] 24

17: Program flowchart for subtractive clustering [30] 36

18: Process of classification....50

19: Several temporal input features of an ECG signal [6] 51

20: RR previous interval (RRp) and RR subsequent interval (RRs) input features [6] 51

21: Amplitude input features [6]....52

22: General ECG block diagram for this thesis 53

23: Method of classification with ANFIS 54

24: General ANFIS structure 55

25: The Pan-Tompkins algorithm for QRS complexes [21] 61

26: One heartbeat showing the results of the WFDB toolbox function 62

27: Histogram of normal P amplitudes 63

28: Classification program flowchart....66

29: RMSE training and checking curves for the first ANFIS under three membership functions for each input for 1000 iterations. Initial step size is 0.01 70

30: RMSE training and checking curves for the first ANFIS under three membership functions for each input for 1000 iterations. Initial step size is 0.001. Convergence was reached at 78 iterations....71

31: Step curve for the first ANFIS under three membership functions for each input for 1000 iterations. Initial step size is0.001....72

32: Initial subtractive clustering FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function. The gray function represents the third membership function....73

33: Adaptive FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function. The gray function represents the third membership function....74

34: Antecedent and consequent parameters before and after training for subtractive clustering FIS. This is the first input's first membership function parameters for the antecendent parameters and the first rule's consequent parameters.....75

35: Decision surface generated from the first ANFIS after 246 iterations between the first input (QRS interval) and the fifth input (RR ratio).....76

36: Trial 1 RMSE training and checking curves for the six ANFIS' (1-6) under three membership....77

37: Trial 2 RMSE training and checking curves for the six ANFIS' (1-6) under three membership functions for each input 81

38: Trial 3 RMSE training and checking curves for the six ANFIS' (1-6) under three membership functions for each input 83

39: Neural network structure for gradient descent and Levenberg-Marquardt algorithms.....87

40: Trial 1 RMSE training and checking curves for the first ANFIS training and checking data for gradient descent. Convergence was reached at 566 iterations from overfitting (minimum checking RMSE reached).....88

41: Trial 1 RMSE training and checking curves for the first ANFIS training and checking data for Levenberg-Marquardt. Convergence was reached at 13 iterations from overfitting (minimum checking RMSE reached).....92

42: Trial 1 RMSE training and checking curves for the first ANFIS under two membership functions grid partitioned for each input for 100 iterations. Initial step size is 0.01. Convergence was reached.....96

43: Trial 1 initial grid partition FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function.....97

44: Trial 1 adapted FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function.....98

45: Antecedent and consequent parameters before and after training for grid partition FIS 99

CHAPTER 1: INTRODUCTION

1.1: Project Goal

Electrocardiograph (ECG) signal tests allow detection of various characteristics in a patient's heart. Characteristics include abnormalities, size and position of chambers, damage to tissue, cardiac pathologies present, and heart rate. The problem with today's ECG signal instruments is the inability to characterize the signals without a doctor's complete evaluation and diagnosis [28]. Research in the field of Computational Intelligence gives promising research results in order to solve this problem. An Adaptive Neuro-Fuzzy Inference System (ANFIS) is a type of neuro-fuzzy classifier and is one of many areas of study in Computational Intelligence (CI). The goal of this project is to explore various applications of an ANFIS to classify well known ECG heartbeats. An additional goal is to compare the ANFIS with artificial neural network (ANN) algorithms.

1.2: Overview

This thesis informs the reader in Chapters 1-4 of theoretical aspects and methodology behind ECG classification. Chapters 5 and 6 present results through simulation and experimentation as well as conclusions and future works. The MATLAB code and a project analysis report can be found in the appendices.

1.3: Software Environment

Simulations and programs for this thesis were programmed through MATLAB® 8.1.0.604 (R2013a) 32-bit version. This environment was selected because of its simplicity in programming and debugging signal processing and matrix-based mathematical operation programs. An extensive number of integrated functions for viewing and analyzing ECG signals as well as optimized matrix math operations greatly accelerated development through MATLAB. This project was made possible through

the MATLAB® NEURAL NETWORK TOOLBOX™ version 8.1.0.604 and MATLAB® FUZZY LOGIC TOOLBOX™ version 2.2.17 additions to the MATLAB student version.

1.4: Research Importance

Cardiac arrhythmias are one of the reasons for high mortality rate. The study of ECG pattern and heart rate variability in terms of computer-based analysis and classification of diseases can be very helpful in diagnostics [15]. This thesis falls under the field of CI. Applications range from adaptive learning to speech recognition. The field draws from biological concepts such as the brain and the physiological decisions of organisms. Breakthroughs in applying CI to medical diagnosis have shown to be successful in the past and are thus important to heart signal classification [29]. This type of data processing could be effectively applied to other biometrics such as electroencephalograms (EEGs) or electromyography (EMGs) to detect the existence of abnormalities in the brain or muscles respectively [2].

The purpose of this study is to aid the cardiologist in diagnosis through neuro fuzzy model classifiers. The rules of neuro fuzzy model classifiers allow for an increase in the interpretability and understandability of the diagnosis. A physician can easily check a fuzzy model classifier for plausibility, and can verify why a certain classification result was obtained for a certain heartbeat by checking the degree of fulfilment of the individual fuzzy rules [7].

1.5: Past Research

Classsifying heartbeats have been performed through an adaptive neuro-fuzzy inference system (ANFIS). An ANFIS utilizes both fuzzy logic and ANNs to approximate the way humans process information through tuning rule-based fuzzy systems [14]. An ANN is a neural network that is a semi-parametric, data-driven model capable of learning complex and highly non-linear mappings [1]. It is to be noted that both the ANFIS and ANN are supervised learning networks. This means a “teacher” must be present in the form of training data in order to train, validate, and test the network. An ANFIS approach to classification between a normal heartbeat and a premature ventricular contraction (PVC) heartbeat has

been studied with the ANFIS producing classification results at a faster convergence than the ANN in addition it allows for human interpretability. The only limitation is its computational complexity. A PVC is an abnormal heartbeat that is characterized as an extra heartbeat.

A decision, whether positive or negative, is exclusively from the ANFIS rule-based structure. Classification between normal and PVC heartbeats achieved an accuracy of 98.53%. [7]. Figure 1 shows the detected PVCs out of normal beats (N), and a graph showing the degree of fulfillment versus the particular rules needed to detect the PVCs. For a more complete analysis of past research in terms of different algorithms of classifying ECG signals, see Chapter 3.

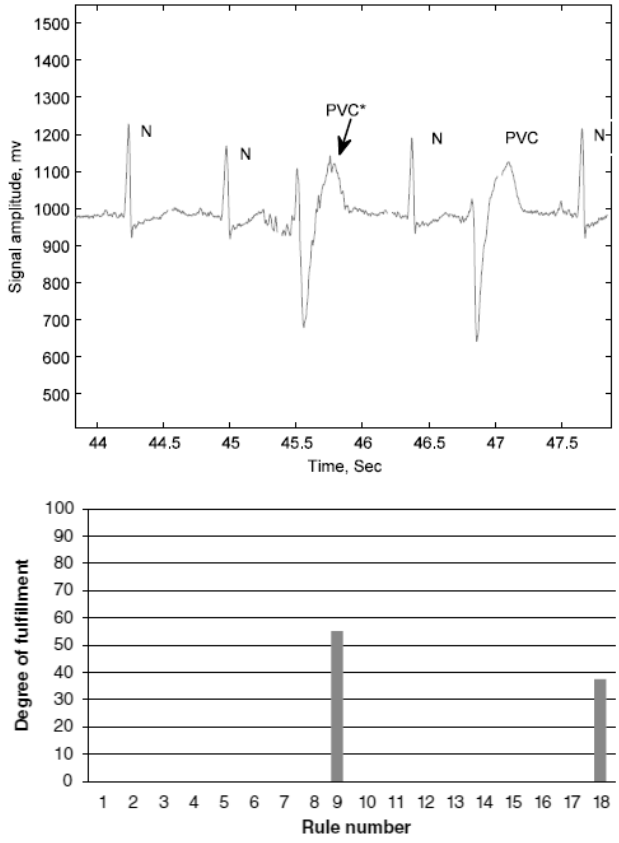


Figure 1: ECG signal showing two PVCs detected and corresponding degree of fulfilment versus ANFIS decision rule [7]

1.6: Brief Approach

For ECG classification, a database of signals is used to observe and extract features. The MIT-BIH (Massachusetts Institute of Technology-Beth Israel Hospital) Arrhythmia Database consists of 48 half-hour excerpts of two-channel ambulatory ECG records which were obtained from 47 subjects studied by the BIH Arrhythmia Laboratory between 1975 and 1979. The sampling rate of the recordings is 360 samples per second. The bit resolution is 11 bits. The amplitude range is 10 mV. Two cardiologists have made the diagnosis for these various records and they have annotated each cardiac cycle. The annotations are important for learning in the neuro-fuzzy classifier [20].

Before classification, the database signals are preprocessed for both observation and training. Preprocessing includes passing the signal through a low-pass filter to remove the 60 Hz power noise for ease of observation of the signal. The given signal would have a baseline shift and would not represent the true amplitude therefore a high pass filter is then used to detrend the signal to direct current (DC) baseline level in order to obtain amplitude information from the signals [19]. The cardiologist's annotations of each heartbeat for each ECG signal are then read from a downloadable MATLAB package from the online database. A pre-made algorithm for detecting the various parts of an ECG signal is then applied to complete the annotation.

ANFIS would classify each heartbeat of an ECG signal. For example, the ANFIS is used here to classify between normal and abnormal heartbeats. It is a binary classifier and thus has one output to the network [15]. The normal and abnormal heartbeats are discussed in Section 2.3. The annotations of each ECG signal allow for inputs to the classifier to be trained. These inputs are different characteristics of an ECG signal. The characteristics are usually temporal intervals and amplitudes of the various parts of the signal. The inputs are then passed through an ANFIS for classification.

To summarize the limitations with traditional methods and advantages over conventional methods for the ANFIS, we begin with convergence speed. It has been shown that the ANFIS has a faster convergence speed than a typical ANN. This is due to a smaller fan-out for backpropagation and the network's first three layers are not fully connected. Smoothness is also guaranteed by interpolation.

Limitations are computational complexity restrictions. This is due to the exponential complexity of the number of rules for grid partitioning. There are surface oscillations around points caused by a high partition of fuzzy rules for grid partitioning. A large number of samples would slow down the subtractive clustering algorithm. Grid partitioning and subtractive clustering are discussed in Section 2.4. [15].

CHAPTER 2: BACKGROUND

2.1: Electrocardiography

Electrocardiography (ECG or EKG) is an approach to measuring the heart's electrical activity. It is typically a non-invasive approach to detecting abnormalities in the heart. Figure 2 shows an ECG signal with various characteristics. A typical heartbeat or cycle begins with a P wave followed by a QRS complex. The beat then ends with a T wave. Occasionally, a U wave appears after the T wave.

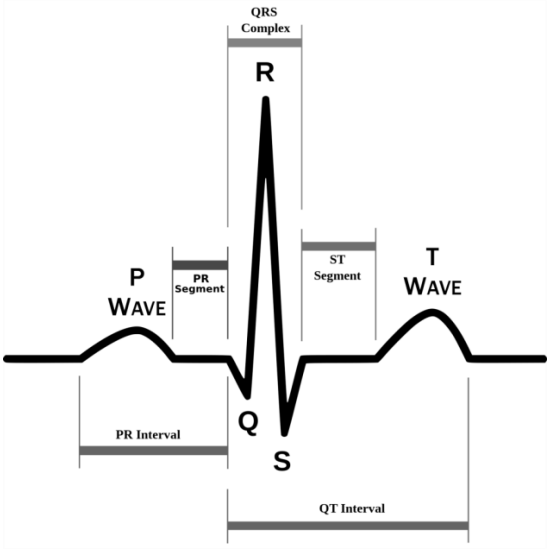


Figure 2: ECG signal [23]

An ECG test can be performed by a local doctor. An ECG measuring device is usually twelve leads. Figure 3 shows this device. ECG interprets the heart's electrical activity through amplitude over time. Ten electrodes are used in a twelve-lead ECG device. Six electrodes are placed across the chest. The remaining four electrodes are placed on the limbs: left arm (LA), right arm (RA), left leg (LL), and right leg (RL).

Figure 3: ECG device

The output of the ECG device is made up of signals from each of the twelve leads printed on a piece of paper. Figure 4 shows an example ECG signal printed on paper. From the paper, the doctor can diagnose the patient. The paper shows limb leads: I, II, III, and augmented leads: aVR, aVL, aVF, and the six chest leads: V1-V6. The three limb electrodes form both the limb leads and augmented leads. Figure 5 shows the placement of the twelve leads

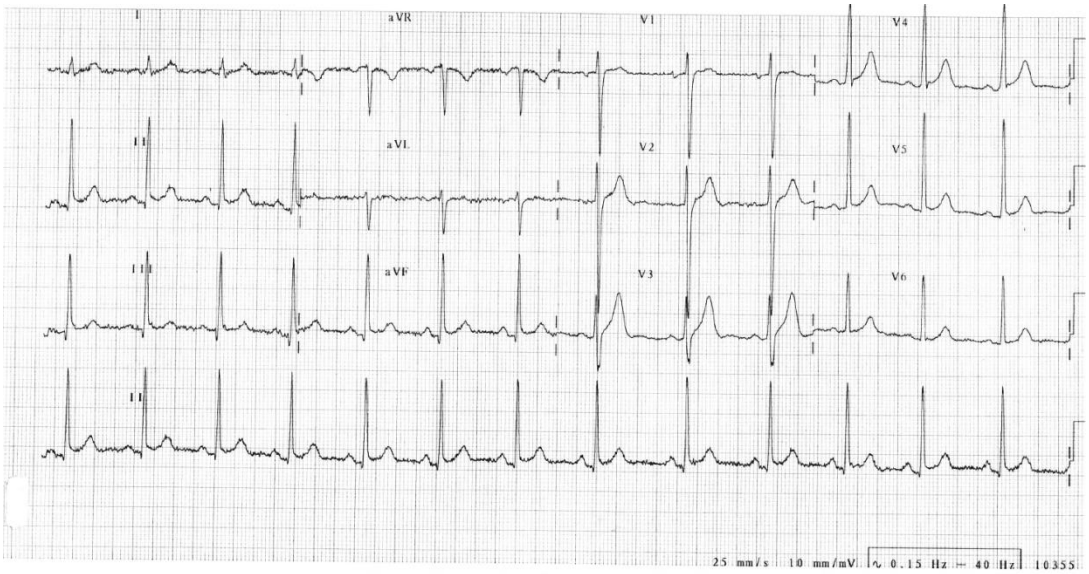


Figure 4: ECG paper

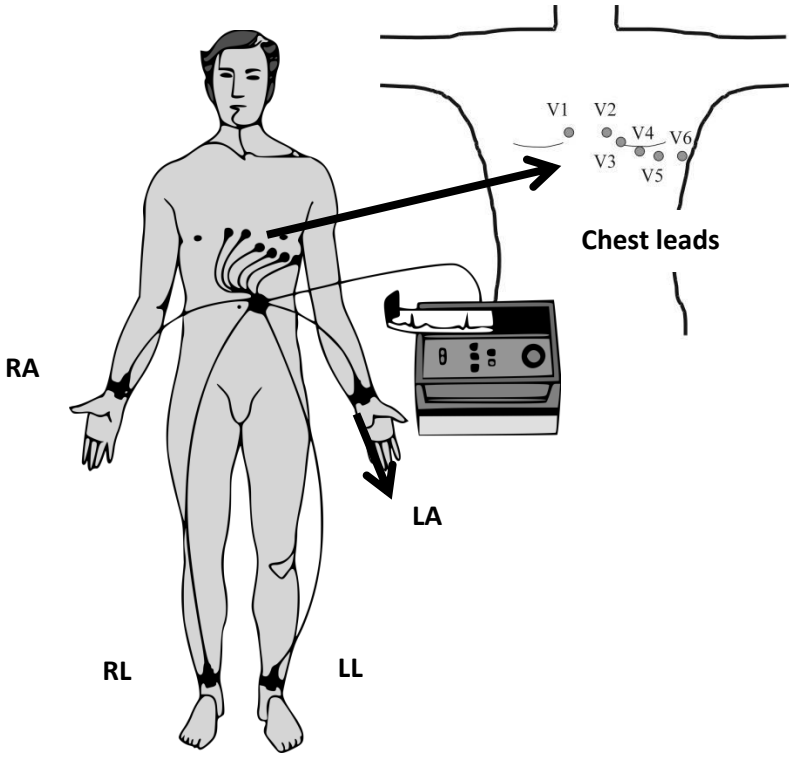


Figure 5: Ten electrodes (Twelve-leads) [9]

Electrocardiography was developed by Willem Einthoven in 1901. He built a string galvanometer to detect low current in order to record the heart's electrical activity. Instead of having an electrical system to perform electrocardiography, he used salt solutions to record results. Over the course of twenty years, the setup went from 600 pounds to 30 pounds. Computers and microelectronics increased the effectiveness of ECG treatment in terms of accuracy and reliability [20].

2.2: The Human Heart

The heart is a muscular organ that pumps blood carrying oxygen and nutrients throughout the body through the circulatory system. Figure 6 shows the human heart. It is divided into four chambers: upper left and right atria; and lower left and right ventricles.

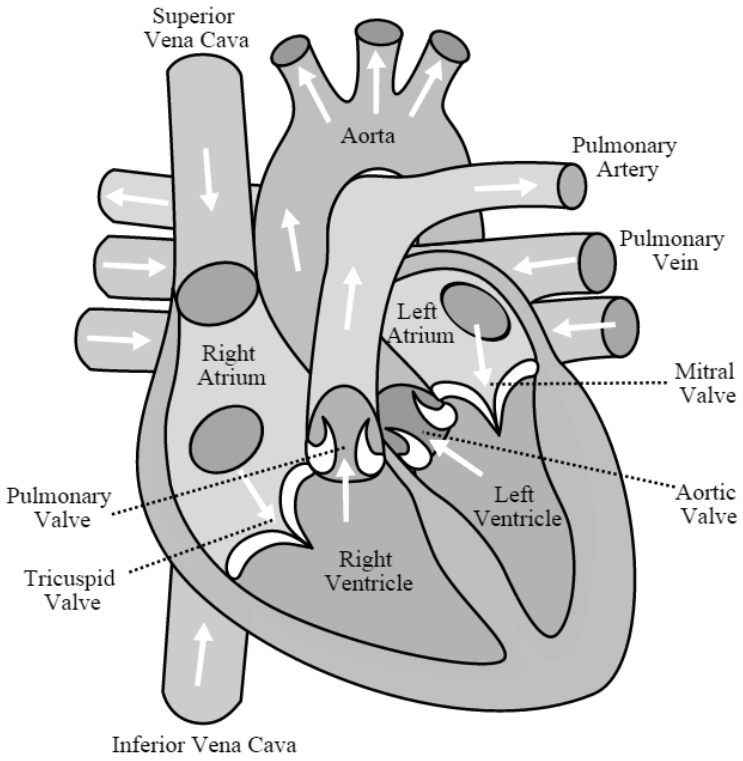


Figure 6: The human heart [27]

Figure 7 shows the human heart in terms of its electrical activity. In a normal ECG heartbeat there are five prominent points. The first is the P wave and corresponds to an atrial depolarization. This happens after the sinoatrial (SA) node generates an impulse. The node is located in the right atrium. The second is the QRS complex and corresponds to atrial repolarization and ventricular depolarization. By this time the ventricular excitation is complete. There is then a ventricular repolarization through the T wave.

The depolarization repolarization phenomena of the heart muscle cells are caused by the movement of ions. This is the essence of the heart electric activity. Movement of ions in the heart muscle cells is the electric current, which generates the electromagnetic field around the heart [8].

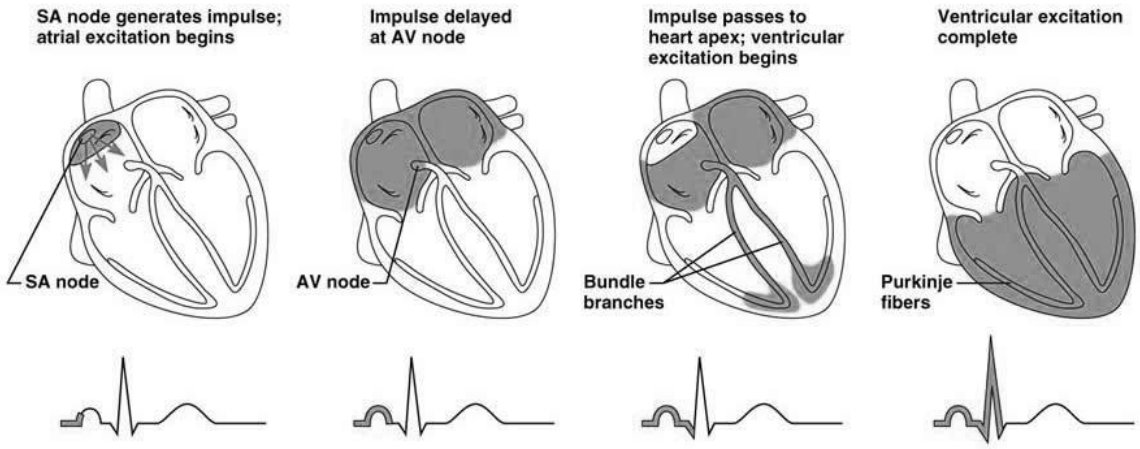


Figure 7: Illustrative ECG relation to the human heart [27]

2.3: Heart Abnormalities

This section introduces five heart abnormalities. These beats are not life threatening but pose an abnormal characteristic that gives way to a disease.

Premature Ventricular Contraction (PVC):

A PVC is an extra heartbeat resulting from abnormal electrical activation originating in the ventricles before a normal heartbeat would occur. This means the purkinje fibers are fired at the ventricles rather than the sinoatrial node. PVCs are usually nonthreatening and are common among older people and patients with sleep disordered breathing. Frequent PVCs are due to physical or emotional stress, intake of caffeine or alcohol, and disorders that cause the ventricles to enlarge such as heart failure and heart valve disorders. Figure 8 shows an example PVC heartbeat.

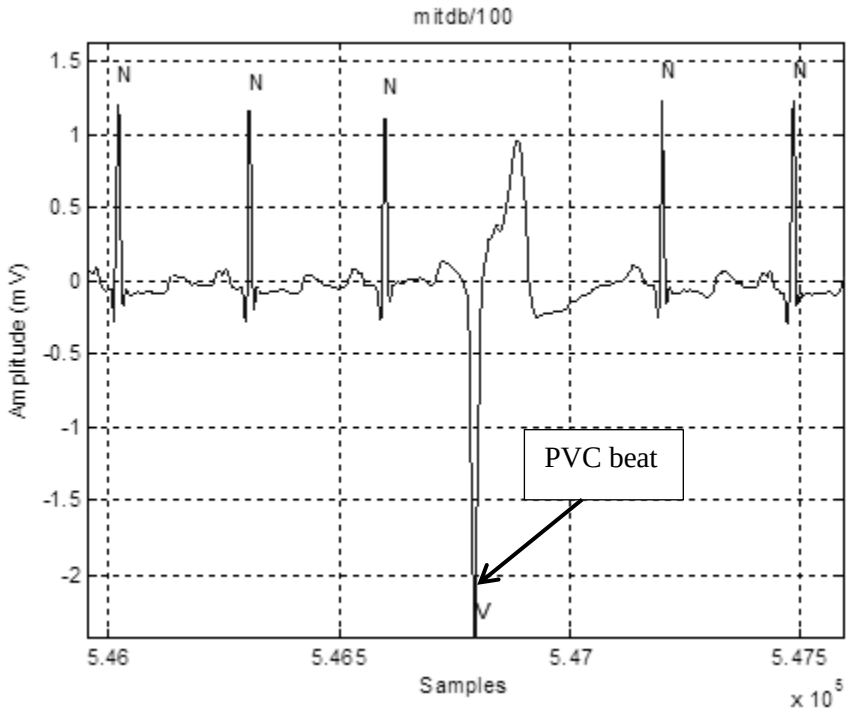


Figure 8: Example of a PVC beat from MIT-BIH database, record 100, lead: Modified Limb Lead II (MLII)

PVCs have the following features [5]:

- Broad QRS complex with duration greater than 120 milliseconds.
- Single-repeated abnormal QRS morphology or multiple abnormal QRS morphology.
- Premature with a compensatory pause (lengthened RR with subsequent heartbeat).
- Either ST depression or T wave inversion in leads with dominant R wave or ST elevation with upright T waves in leads with a dominant S wave.

Atrial Premature Contraction (APC):

An APC is an extra heartbeat resulting from abnormal electrical activation originating in the atria before a normal heartbeat would occur. This means the purkinje fibers are fired at the atria rather than the sinoatrial node. APCs are usually nonthreatening and are common among heathy young and elderly people. It can be perceived as a skipped heartbeat. Frequent APCs are mainly due to physical or emotional stress and intake of caffeine or alcohol. Figure 9 shows an example APC beat.

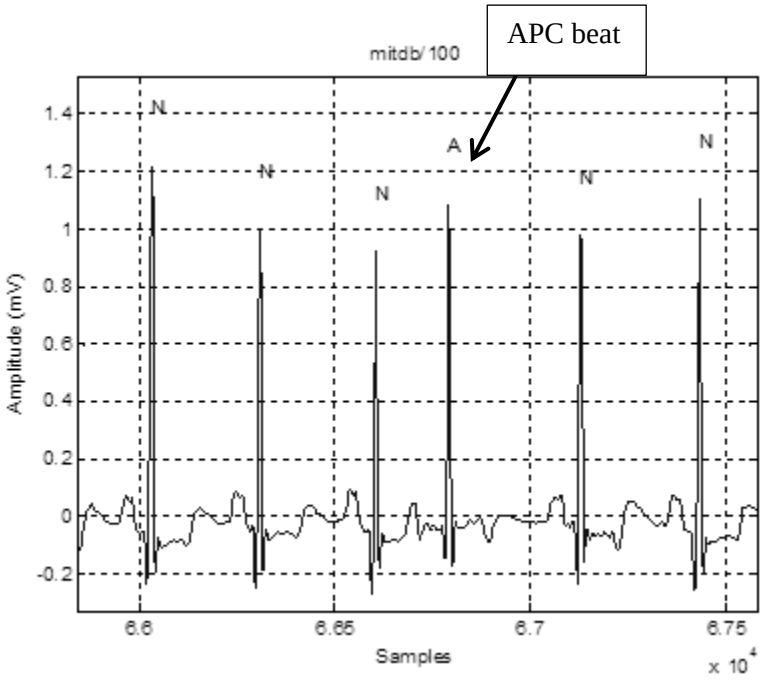


Figure 9: Example of an APC beat from MIT-BIH database, record 100, lead: Modified Limb Lead II (MLII)

APCs have the following features [5]:

- Narrow QRS complex with duration less than 120 milliseconds.
- Shortened RR interval with previous heartbeat.
- Premature with a compensatory pause (lengthened RR interval with subsequent heartbeat).

Left Bundle Branch Block (LBBB):

A LBBB is a condition where the left ventricle contracts later than the right ventricle due to delayed activation of the left ventricle. Frequent LBBBs are mainly due to hypertension, inadequate blood supply to the heart muscles, and heart valve diseases. Figure 10 shows an example LBBB heartbeat.

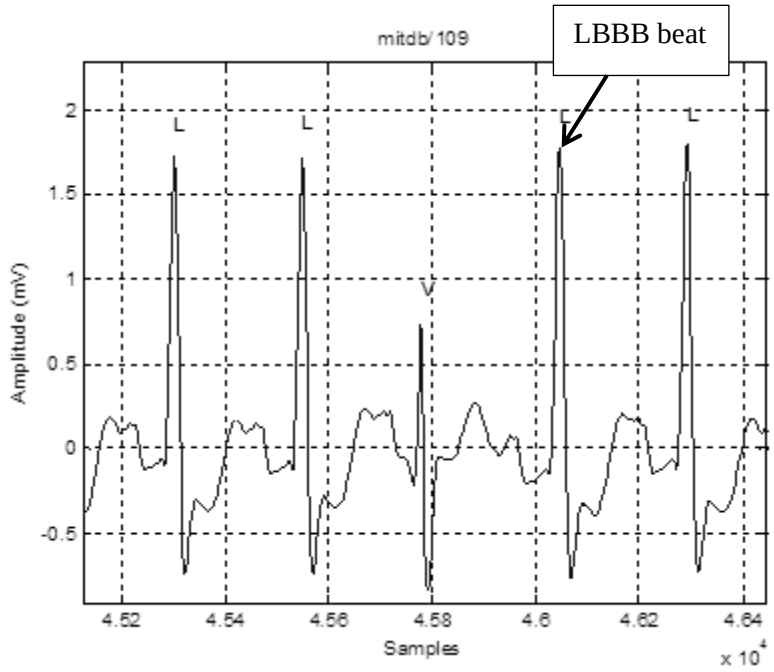


Figure 10: Example of a LBBB from MIT-BIH database, record 109, lead: Modified Limb Lead II (MLII)

LBBBs have the following features [5]:

- Broad QRS complex with duration greater than 120 milliseconds.
- ST wave is deflected opposite of the QRS complex.
- ST segments and T waves always go in the opposite direction to the main vector of the QRS complex.

Right Bundle Branch Block (RBBB):

A RBBB is a condition where the right ventricle does not completely contract in the right bundle branch. The left ventricle contracts normally in the left bundle branch. Frequent RBBBs are mainly due to enlargement of the tissue in the right ventricles, inadequate blood supply to the heart muscles, and heart valve diseases. Figure 11 shows an example RBBB beat.

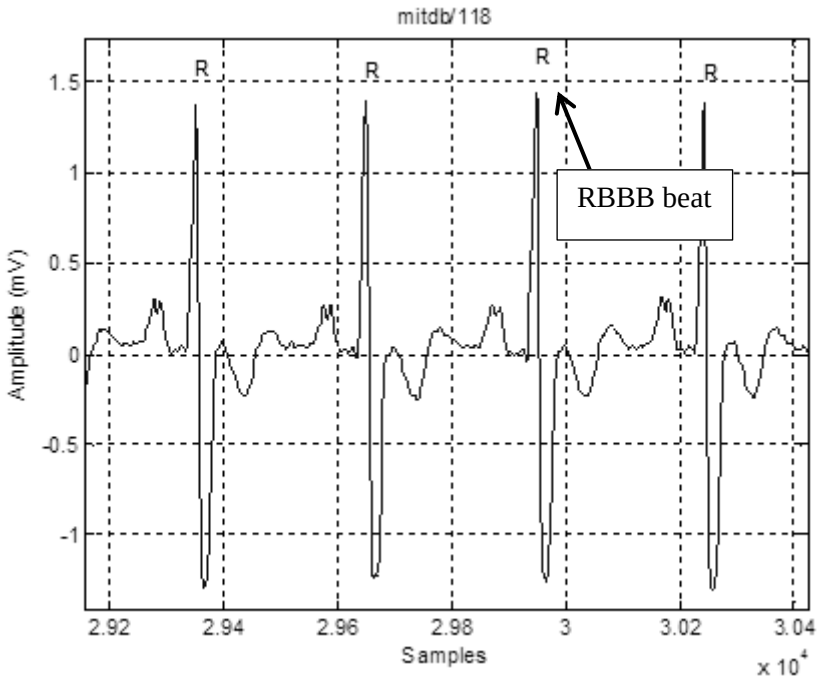


Figure 11: Example of a RBBB from MIT-BIH database, record 118, lead: Modified Limb Lead II (MLII)

RBBBs have the following features [5]:

- Broad QRS complex with duration greater than 120 milliseconds.
- Slurred S waves.
- T waves deflected opposite the terminal deflection of the QRS complex.
- ST depression and T wave inversion in the right precordial leads (V1-3)

Paced beat:

A paced beat is the result of a patient with an artificial pacemaker. Figure 12 shows an example of a paced heartbeat.

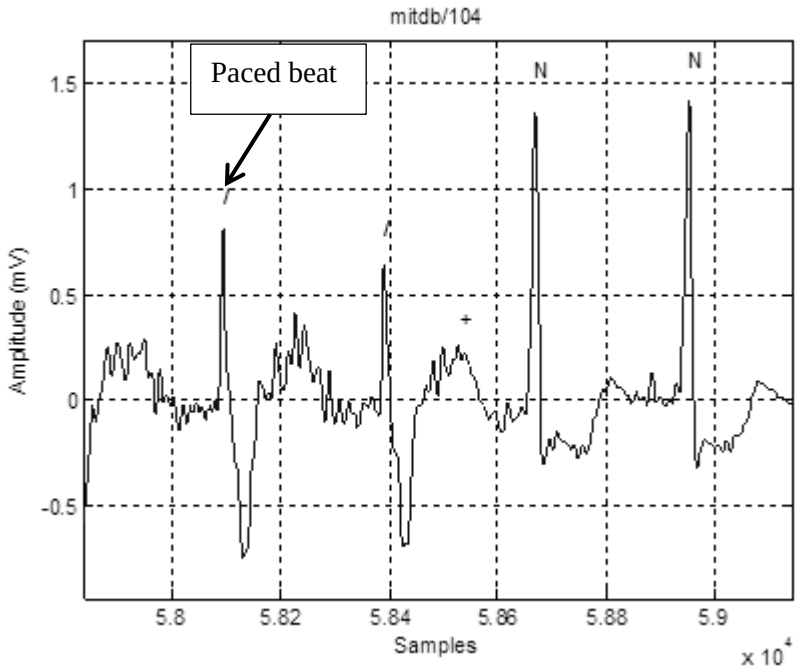


Figure 12: Example of a paced beat from MIT-BIH database, record 104, lead: Modified Limb Lead II (MLII)

A paced heartbeat has the following features [7]:

- Vertical spikes of short duration (usually 2 milliseconds) before either the P or Q waves.
- Morphology on P or Q waves.
- Prolonged PR interval (280 milliseconds).
- Morphology on QRS complex. QRS complex resembles a ventricular heartbeatbeat.

2.4: Fuzzy Set Theory

Fuzzy sets provide a framework for incorporating human knowledge in the solution of problems. It is the basis of the ANFIS. In Fuzzy Logic theory, sets are associated with set membership. Compared to the traditional binary sets or "crisp sets" where membership is either '1' typically indicating true or '0' indicating false, fuzzy logic variables ranges between 0 and 1. Thus, Fuzzy Logic deals with approximate reasoning rather than fixed and exact reasoning [12].

The theory can be defined as follows. Let *Z* be a set of elements that is equivalent to *z*: $Z = \{z\}$ Let *A* be a fuzzy set in *Z* characterized by a membership function, $\mu_A(z)$ be a membership function that associates with each element of *Z* a real number in the interval [0, 1]. The elements, *z*, can be considered full, partial, or no membership to their corresponding membership functions. This can be expressed as $A = \{z, \mu_A(z) \mid z \in Z\}$. Set theory can be applied to fuzzy sets. There is the possibility of empty sets, equivalent sets, complements, subsets, unions, or intersections. Common membership functions are gaussian, gaussian-bell, triangle, and trapezoidal shown in Figure 13 [12].

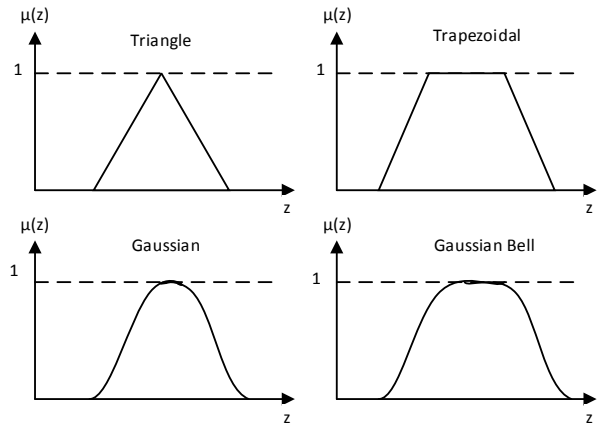


Figure 13: Several membership functions μ to choose from for a fuzzy set Z

A Fuzzy Inference System (FIS) formulates the mapping from a given input to an output using fuzzy sets. Figure 14 shows a FIS block diagram. The system comprises of five steps:

1. Fuzzification of the input variables
2. Application of the fuzzy operator (AND or OR) in the antecedent
3. Implication from the antecedent to the consequent
4. Aggregation of the consequents across the rules
5. Defuzzification

In Fuzzification, the inputs are mapped by membership functions to determine the degree to which the inputs belong.

In applying a fuzzy operator, the result of the antecedent and consequent of an if-then rule is found. From an antecedent or consequent, the fuzzy membership values can be found. An AND operator would take the minimum of the two limits of the fuzzy membership values. An OR operator would take the maximum of the two limits of the fuzzy membership values.

Implication from the antecedent to the consequent involves assigning weights to each rule. The consequent is a fuzzy set represented by a membership function, which weights appropriately the linguistic characteristics that are attributed to it. The consequent is reshaped using a function that either

truncates or scales the output fuzzy set. The input of the implication process is a single number given by the antecedent. The output is a fuzzy set. Implication is then implemented for each rule.

Next, aggregating all outputs of each rule into a single fuzzy set is done. Note this step should be commutative so that the order in which the rules are executed is unimportant. Three methods of aggregations can be applied: maximum, probabilistic OR, or simply the sum of each rule's output set.

Defuzzification generates a single number from the aggregation step. The most popular method is the centroid calculation, which returns the center of area under the aggregate output curve. Other methods are bisector and average of certain ranges of the aggregate output curve [26].

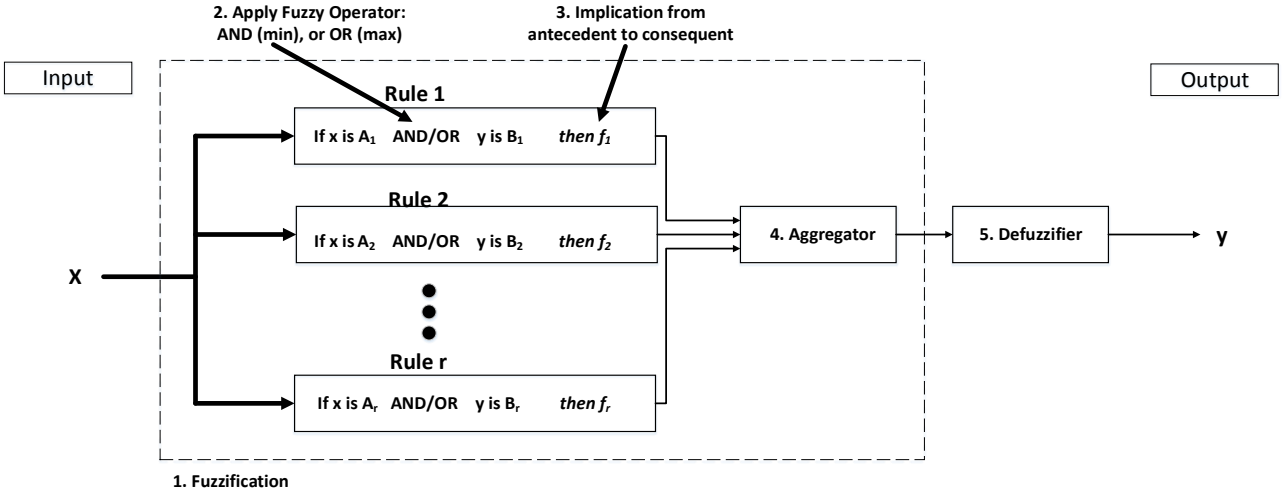


Figure 14: FIS diagram [12]

There are several fuzzy models for FIS. Two prominent models are the Mamdami and Sugeno models. The FIS discussed previously adheres to a Mamdami model. The Sugeno model however, determines the output not as a membership function, but as a constant or linear term.

The Sugeno fuzzy model is described as follows. Let $x = [x_1, x_2, ..., x_n]^T$ be the input vector and o_i denote the output (consequent). Let R_i denote the *i*th rule, and $A_{i1}, ..., A_{in}$ are fuzzy sets defined in the antecedent space by membership functions $\mu_{Aij}(x_j)$ defined for all real numbers between '0' and '1'. Let

$p_{i1},..., p_{i(n+1)}$ represent the consequent parameters and M is the number of rules. The set of rules and the rule consequents are linear functions defined as:

$$R_i : \textbf{if } x_1 \text{ is } A_{i1} \textbf{ and ... } x_n \text{ is } A_{in} \textbf{ then}$$
$$o_i = p_{i1}x_1 + \ldots, p_{in}x_n + p_{i(n+1)}, \; i = 1, ..., M \tag{1}$$

Defuzzification in the model is defined by a hyperplane in the antecedent-consequent product space. Let β_i denote the degree of fulfillment of the *i*th rule:

$$\beta_i = \prod_{j=1}^{n} A_{ij}(x_j), \; i = 1, ..., M \tag{2}$$

The output y of the model is computed through the center of gravity of the final fuzzy set [7]:

$$y = \frac{\sum_{i=1}^{M} \beta_i o_i}{\sum_{i=1}^{M} \beta_i} \tag{3}$$

There are several partitioning methods for FIS input spaces to form the antecedents of fuzzy rules. The grid partition consists of dividing each input variable domain into a given number of intervals whose limits do not necessarily have any physical meaning and do not take into account a data density repartition function [12]. Another is Subtractive clustering. This is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers in a set of data. A data point with the highest potential is selected as a cluster center. Data points in the vicinity of the cluster center are removed to determine the next data cluster and center location. The process iterates until it is within radii of a cluster center. Section 2.5 discusses the ANFIS algorithm assuming grid partition. Section 2.8 discusses how the subtractive clustering can be used in conjunction with the ANFIS as an improvement instead of the grid partition in terms of rule reduction and overall classification of heartbeats. Both grid partition and subtractive clustering are simulated. Grid partition results are discussed in Section 5.4. Subtractive clustering results are discussed in Section 5.1 [26].

2.5: ANFIS Algorithm

An ANFIS, as mentioned before, combines the FIS with neural networks to tune the rule-based fuzzy systems. Two common FIS structures can then be applied: Sugeno or Mamdani. The Sugeno method is chosen in this thesis because it is computationally efficient, works well with linear techniques, works well with optimization and adaptive techniques, and it is well suited to mathematical analysis. The advantage of Mamdani is that it's intuitive and well suited to human input. The disadvantage of Mamdani is it's computationally expensive because another set of parameters is added to increase human interpretablity.

The Sugeno ANFIS has the premise part of the fuzzy rule as a fuzzy proposition and the conclusion part as a linear function. There are five layers of the structure discussed below and shown in Figure 15. A rectangle represents an adaptive node. Assuming a Sugeno fuzzy model, fuzzy-if-then rules are applied. This is shown in Figure 16. A first-order Sugeno ANFIS structure is used in order to output a linear function. A zero-order Sugeno ANFIS structure would output a constant parameter.

Parameters are calculated by the hybrid learning algorithm. Let S_1 represent the antecedent (nonlinear) parameters and S_2 represent the consequent (linear) parameters. The hybrid learning algorithm updates both parameters. This algorithm is discussed in Section 2.6.

Let n represent the number of inputs. Let j represent the layers of the ANFIS where j = {1, 2, 3, 4, 5}. Let output of node i of layers j be O_{ji}. Let p represent the number of membership functions of each input.

Layer 1: This is the fuzzification layer. The output of this layer is O_{1i} where $i = \{1, 2, ...p \cdot n\}$. O_{1i} is a membership function that specifies the degree to which the given input satisfies the fuzzy sets A_k for $k = \{1, 2, ...p\}$. The fuzzy sets are represented as membership functions. The functions are expressed as $\mu_{A_k}(x_t; \{a, b, c, d, ...\})$ for $t = \{1, 2, ... n\}$ where the input n features are grid partitioned into p membership functions. Each membership function is a function of the feature x_m.

$$O_{1i} = \mu_{A_k}(x_m; \{a_i, b_i, c_i, d_i, ...\}) \tag{4}$$

The membership functions represent the antecedent parameters of the ANFIS described as $S_{1i} = \{a_i, b_i, c_i, ...\}$. The new expression for the output O_{1i} is as follows:

$$O_{1i} = \mu_{A_k}(x_m; S_{1i}) \tag{5}$$

The nodes of this layer are adaptive. See Section 2.6 for details on adaptation of the antecedent parameters. There are several membership functions of fuzzy element z in the Sugeno structure that consist of antecedent parameters. They are listed as follows:

$$\textbf{Gaussian: } \mu(z; a, b) = e^{-\frac{1}{2}(\frac{z-a}{b})^2} \tag{6}$$

Where a represents the center and b represents the function's width.

$$\textbf{Gaussian-bell: } \mu(z; a, b, c) = \frac{1}{1+|\frac{z-a}{b}|^{2c}} \tag{7}$$

Where a represents the center, b represents the function's width, and c represents both the direction of the bell and its width.

$$\textbf{Triangle: } \mu(z; a, b, c) = \begin{cases} 0, & z \le a \\ \frac{z-a}{b-a}, & a \le z \le b \\ \frac{c-z}{c-b}, & b \le z \le c \\ 0, & c \le z \end{cases} \tag{8}$$

Where $a \le b \le c$ and $\{a,b,c\}$ represents the z-coordinates of the three corners of the underlying triangle.

$$\textbf{Trapezoid: } \mu(z; a, b, c, d) = \begin{cases} 0, & z \le a \\ \frac{z-a}{b-a}, & a \le z \le b \\ 1, & b \le z \le c \\ \frac{d-z}{d-c}, & c \le z \le d \\ 0, & d \le z \end{cases} \tag{9}$$

Where $a \le b \le c \le d$ and $\{a, b, c, d\}$ represents the z-coordinates of the four corners of the underlying trapezoid.

Let l represent the number of antecedent parameters for each membership function $\mu(z)$. The total number of antecedent parameters are then equivalent to:

$$\text{Total number of antecedent parameters} = l \cdot (p \cdot n) \tag{10}$$

Layer 2: This is the rule layer. The output of this layer is O_{2i} where node $i = \{1, 2, ...p^n\}$. O_{2i} are rules that are defined either by an AND (minimum of incoming signals) or an OR (maximum of incoming signals). Let R represent the rule choice of the second layer nodes in Figure 15.

$$R = \{\min[AND]\} \text{ or } R = \{\max[OR]\} \tag{11}$$

Let W_i represent the weights from the rule nodes.

$$O_{2i} = W_i = rule\{A_k\} \tag{12}$$

Layer 3: This is the normalization layer. Let N represent the normalization of the nodes in layer 3 of Figure 15. The output of this layer is O_{3i} where node $i = \{1, 2, ...p^n\}$. Let $\overline{W_i}$ represent the normalized weight of each rule.

$$O_{3i} = \overline{W_i} = \frac{W_i}{W_1 + W_2 + ...W_i} \tag{13}$$

Normalizing guarantees stable convergence of weights and biases. It also avoids the time-consuming process of defuzzification.

Layer 4: This is the defuzzification layer. The output of this layer is O_{4i} where node $i = \{1, 2, ...p^n\}$. Let $S_{2i} = \{q_{1i}, q_{2i}, ...q_{np^n}, r_i\}$ be the consequent parameters. A linear function f_i is expressed as the multiplication of the inputs with the corresponding consequent parameters:

$$f_i = q_{1i}x_1 + q_{2i}x_2 + ... q_{np^n}x_n + r_i \tag{14}$$

The output O_{4i} is the product of the normalized firing strength $\overline{W_i}$ with the linear function f_i:

$$O_{4i} = \bar{W}_i \; f_i = \bar{W}_i (q_{1i} x_1 + q_{2i} x_2 + \ldots q_{np^n} x_n + r_i) \tag{15}$$

$$\text{Total number of consequent parameters} = (n+1) \cdot p^n \tag{16}$$

Layer 5: This is the summation layer. The output of this layer is O_{5i} where node $i = \{1\}$. Since there is only one output, the ANFIS is a binary classifier. The output is the aggregation of all defuzzified outputs O_{4i} from layer 4, and thus it follows the center of gravity equation (3):

$$O_{51} = \sum_i O_{4i} = \sum_i \bar{W}_i \; f_i = \frac{\sum_i W_i f_i}{\sum_i W_i} \text{ for } i = \{1, 2, \ldots p^n\} \tag{17}$$

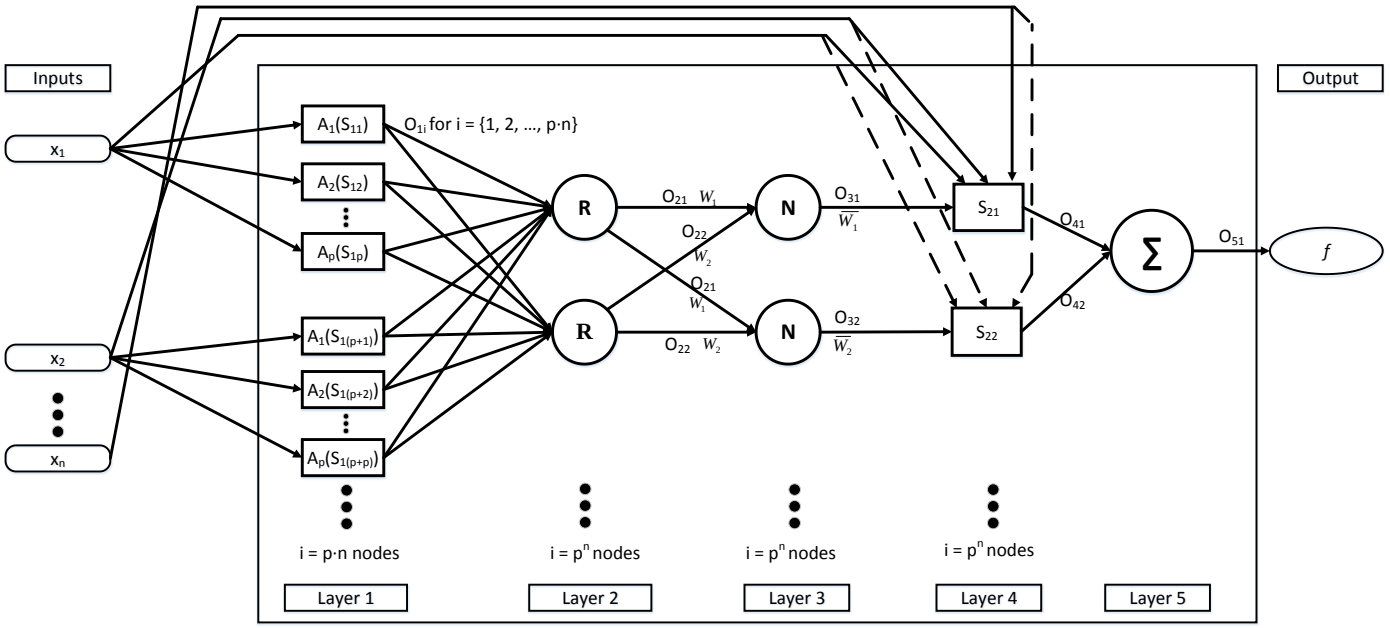


Figure 15: n-input first-order Sugeno Adaptive Neuro-Fuzzy Inference System (ANFIS) [12]

An illustrated example of a two-input, two-rule, first order Sugeno ANFIS model is shown in Figure 16.

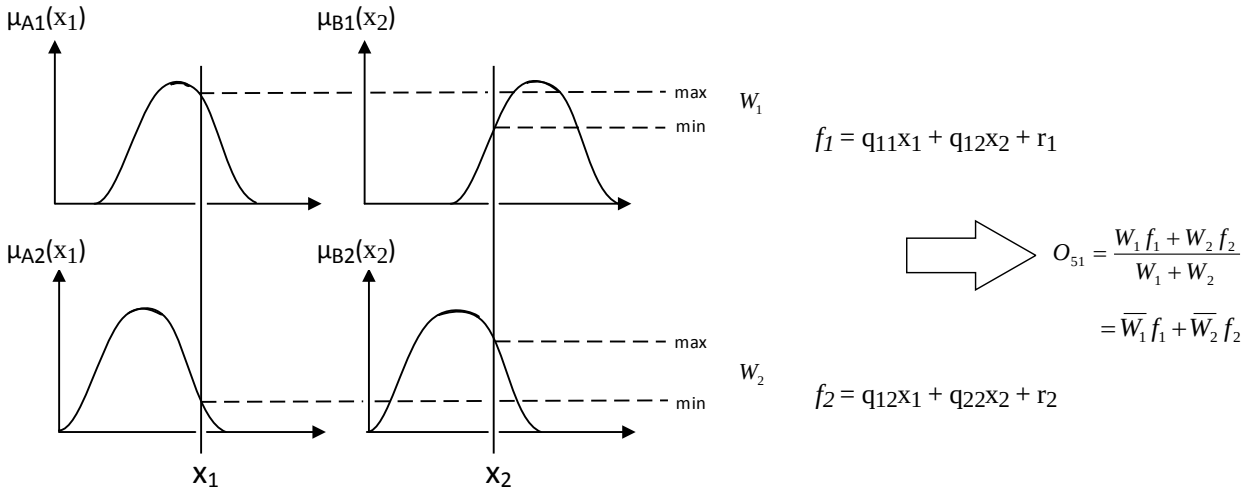


Figure 16: Two-input, two-rule, first order Sugeno ANFIS model [12]

2.6: Hybrid Learning Algorithm

Determination of optimal values of ANFIS parameters is made from a hybrid learning algorithm as discussed in [15]. It combines the least-squares estimator (LSE) method and the backpropagation gradient descent method for training. In the forward pass, the antecedent parameters are assumed fixed while the consequent parameters are identified by the LSE algorithm. In the backward pass, the consequent parameters are assumed fixed while the antecedent parameters are identified by the backpropagation algorithm through gradient descent. This is described in Table 1. This algorithm could be made for both online and offline learning. However, this thesis uses the offline learning approach. As discussed in [37], the advantage of this algorithm is overall optimization of the consequent parameters for the given antecedent parameters. In this way we can, not only reduce the number of dimensions used in the gradient descent algorithm, but also accelerate the rate of convergence of the parameters.

Table 1: Hybrid learning applied to ANFIS [15]

	Forward pass	Backward pass
Antecedent parameters (Non-linear)	Fixed	Gradient Descent
Consequent parameters (Linear)	LSE	Fixed
Signals	Node outputs	Error signals

Let S_1 represent the set of the antecedent parameters and S_2 represent the set of consequent parameters as discussed in Section 2.5. Following equation (17), we can develop an expression involving the normalized weights $\overline{\overline{W}}_i$ multiplied by the inputs x_t :

$$O_{51} = \sum_i [\overline{\overline{W}}_i x_1) q_{1i} + (\overline{\overline{W}}_i x_2) q_{2i} + \ldots (\overline{\overline{W}}_i x_n) q_{ni} + \overline{\overline{W}}_i r_i] \quad \text{for } i = \{1, 2, \ldots p^n\} \tag{18}$$

As discussed in [16], we can represent equation (18) as:

$$y = X_1 f_1(\mathbf{u}) + X_2 f_2(\mathbf{u}) + \cdots + X_j f_j(\mathbf{u}) \tag{19}$$

To identify the unknown consequent parameters X_l, usually a set of iterations (experiments) are performed to obtain a training data set composed of data pairs $\{(\mathbf{u}_l : y_l), l = 1, \cdots, m\}$. Subsitiuting each data pair into equation (19) yields a set of m linear equations:

$$\begin{aligned} f_1(\mathbf{u}_1)X_1 + f_2(\mathbf{u}_1)X_2 + \cdots + f_j(\mathbf{u}_1)X_j &= y_1 \\ f_1(\mathbf{u}_2)X_1 + f_2(\mathbf{u}_2)X_2 + \cdots + f_j(\mathbf{u}_2)X_j &= y_2 \\ \vdots \qquad\qquad & \vdots\vdots \\ f_1(\mathbf{u}_m)X_1 + f_2(\mathbf{u}_m)X_2 + \cdots + f_j(\mathbf{u}_m)X_j &= y_m \end{aligned} \tag{20}$$

Let m and j represent the number of training data and consequent parameters respectively. Let $\mathbf{A}$ represent a $m\ x\ j$ matrix of weights multiplied by inputs as represented in equation (18). This can be represented as a design matrix denoted as:

$$\mathbf{A} = \begin{bmatrix} f_1(\mathbf{u}_1) & \cdots & f_j(\mathbf{u}_1) \\ \vdots & \vdots & \vdots \\ f_1(\mathbf{u}_m) & \cdots & f_j(\mathbf{u}_m) \end{bmatrix} \tag{21}$$

Let $\mathbf{X}$ represent a $j\ x\ 1$ vector of the unknown consequent parameters (assuming one rule $i = 1$) be expressed as:

$$\mathbf{X} = \begin{bmatrix} X_1 = q_{11} \\ \vdots \\ X_{j-1} = q_{n1} \\ X_j = r_1 \end{bmatrix} \tag{22}$$

and $\mathbf{y}$ represent a $m\ x\ 1$ output vector and equivalent as O_{51} for m iterations:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \tag{23}$$

Let the l th row of the joint data matrix $[\mathbf{A} \vdots \mathbf{y}]$ be denoted by $[\mathbf{a}_l^T \vdots \mathbf{y}_l]$ is related to the l th input-output data pair through:

$$\mathbf{a}_l^T = [f_1(\mathbf{u}_l), \cdots, f_j(\mathbf{u}_l)] \tag{24}$$

If $\mathbf{A}$ is square ($m = j$) and nonsingular, then we can solve for $\mathbf{X}$ in the following equation:

$$\mathbf{AX} = \mathbf{y} \tag{25}$$

To obtain:

$$\mathbf{X} = \mathbf{A}^{-1}\mathbf{y} \tag{26}$$

However, usually this is an overdetermined solution ($m > j$) and thus the data might be contaminated with noise. Thus an error vector $\mathbf{e}$ is used to account for this random noise and is expressed as:

$$\mathbf{AX} + \mathbf{e} = \mathbf{y} \tag{27}$$

A search for a minimum parameter (LSE) vector $\mathbf{X} = \mathbf{X}^*$ is made through a sum of squared error as discussed in [16]:

$$E(\mathbf{X}) = \sum_{l=1}^{m} (y_l - \mathbf{a}_l^T \mathbf{X})^2 = (\mathbf{y} - \mathbf{AX})^T (\mathbf{y} - \mathbf{AX}) \tag{28}$$

The least squares estimator $\mathbf{X}^*$ is found when the squared error is minimized satisfying the following equation:

$$\mathbf{A}^T \mathbf{A}\mathbf{X}^* = \mathbf{A}^T \mathbf{y} \tag{29}$$

If $\mathbf{A}^T \mathbf{A}$ is nonsingular and $\mathbf{X}^*$ is unique, then the LSE is:

$$\mathbf{X}^* = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y} \tag{30}$$

If $\mathbf{A}^T \mathbf{A}$ is singular, a sequential method of LSE is used. The sequential method of the LSE is the recursive least squares estimate and is used in this thesis. Here we assume the row dimensions of $\mathbf{A}$ and

$\mathbf{y}$ are k, which represents a measure of time if the data pairs become available in sequential order. This means $\mathbf{X}_k$ is used to calculate $\mathbf{X}_{k+1}$:

$$\mathbf{X}_k = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} \tag{31}$$

$$\mathbf{X}_{k+1} = \left(\begin{bmatrix}\mathbf{A}\\ \mathbf{a}^T\end{bmatrix}^T\begin{bmatrix}\mathbf{A}\\ \mathbf{a}^T\end{bmatrix}\right)^{-1}\begin{bmatrix}\mathbf{A}\\ \mathbf{a}^T\end{bmatrix}^T\begin{bmatrix}\mathbf{y}\\ y\end{bmatrix} \tag{32}$$

For simplification, we let $m\ x\ j$ matrices $\mathbf{P}_k$ and $\mathbf{P}_{k+1}$ be defined as:

$$\mathbf{P}_k = (\mathbf{A}^T\mathbf{A})^{-1} \tag{33}$$

$$\mathbf{P}_{k+1} = \left(\begin{bmatrix}\mathbf{A}\\ \mathbf{a}^T\end{bmatrix}^T\begin{bmatrix}\mathbf{A}\\ \mathbf{a}^T\end{bmatrix}\right)^{-1} = (\mathbf{A}^T\mathbf{A} + \mathbf{a}\mathbf{a}^T)^{-1} \tag{34}$$

The two matrices can be expressed as:

$$\mathbf{P}_k^{-1} = \mathbf{P}_{k+1}^{-1} - \mathbf{a}\mathbf{a}^T \tag{35}$$

$\mathbf{X}_k$ and $\mathbf{X}_{k+1}$ can be expressed as:

$$\begin{aligned}\mathbf{X}_k &= \mathbf{P}_k\mathbf{A}^T\mathbf{y}\\ \mathbf{X}_{k+1} &= \mathbf{P}_{k+1}(\mathbf{A}^T\mathbf{y} + \mathbf{a}y)\end{aligned} \tag{36}$$

To express $\mathbf{X}_{k+1}$ in terms of $\mathbf{X}_k$, an elimination of $\mathbf{A}^T\mathbf{y}$ is made in equation (37):

$$\mathbf{A}^T\mathbf{y} = \mathbf{P}_k^{-1}\mathbf{X}_k \tag{37}$$

Equation (37) is then plugged into equation (36) to obtain:

$$\begin{aligned}\mathbf{X}_{k+1} &= \mathbf{P}_{k+1}(\mathbf{P}_k^{-1}\mathbf{X}_n + \mathbf{a}y)\\ &= \mathbf{P}_{k+1}[(\mathbf{P}_{k+1}^{-1} + \mathbf{a}\mathbf{a}^T)\mathbf{X}_k + \mathbf{a}y]\\ &= \mathbf{X}_k + \mathbf{P}_{k+1}\mathbf{a}(y - \mathbf{a}^T\mathbf{X}_k)\end{aligned} \tag{38}$$

However, calculating $\mathbf{P}_{k+1}$ involves an inversion calculation and is computationally expensive as explained in [16]. From equation (35), we have:

$$\mathbf{P}_{k+1} = (\mathbf{P}_k^{-1} + \mathbf{a}\mathbf{a}^T)^{-1} \tag{39}$$

A matrix inversion formula in [16] can be applied:

$$(\mathbf{T}+\mathbf{BC})^{-1}=\mathbf{T}^{-1}-\mathbf{T}^{-1}\mathbf{B}(\mathbf{I}+\mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CT}^{-1} \tag{40}$$

Let $\mathbf{T}$ and $\mathbf{I}+\mathbf{CT}^{-1}\mathbf{B}$ be nonsingular square matrices and let $\mathbf{T}=\mathbf{P}_k^{-1}$, $\mathbf{B}=\mathbf{a}$, and $\mathbf{C}=\mathbf{a}^T$.

Then $\mathbf{P}_{k+1}$ can be expressed as:

$$\begin{aligned}\mathbf{P}_{k+1}&=\mathbf{P}_k-\mathbf{P}_k a(\mathbf{I}+\mathbf{a}^T\mathbf{P}_k\mathbf{a})^{-1}\mathbf{a}^T\mathbf{P}_k\\&=\mathbf{P}_k-\frac{\mathbf{P}_k\mathbf{a}\mathbf{a}^T\mathbf{P}_k}{1+\mathbf{a}^T\mathbf{P}_k\mathbf{a}}\end{aligned} \tag{41}$$

In summary, the recursive LSE for equation (25) where the k th $(1\le k\le m)$ row of $[\mathbf{A}:\mathbf{y}]$ denoted by $[\mathbf{a}_k^T:y_k]$ is sequentially obtained and is calculated as:

$$\begin{aligned}\mathbf{P}_{k+1}&=\mathbf{P}_k-\frac{\mathbf{P}_k\mathbf{a}_{k+1}\mathbf{a}_{k+1}^T\mathbf{P}_k}{1+\mathbf{a}_{k+1}^T\mathbf{P}_k\mathbf{a}_{k+1}}\\\mathbf{X}_{k+1}&=\mathbf{X}_k+\mathbf{P}_{k+1}\mathbf{a}_{k+1}(y_{k+1}-\mathbf{a}_{i+1}^T\mathbf{X}_k)\end{aligned} \tag{42}$$

where k ranges from 0 to m-1 and the overall LSE $\mathbf{X}^*$ is equal to $\mathbf{X}_m$ (the consequent parameters S_2 for all i rules in the ANFIS), the estimator using m data pairs. In order to initialize the algorithm in equation (43), initial values of $\mathbf{X}_0$ and $\mathbf{P}_0$ is set. In practice [16] and in the thesis, $\mathbf{X}_0$ is set to a zero matrix for convenience and $\mathbf{P}_0$ is expressd as:

$$\mathbf{P}_0=\alpha\mathbf{I} \tag{43}$$

α is set to a positive large number to fulfill the following condition as explained in [16]:

$$\lim_{\alpha->\infty}\mathbf{P}_0^{-1}=\lim_{\alpha->\infty}\frac{1}{\alpha}\mathbf{I}=0 \tag{44}$$

The gradient descent backpropagation can be applied once the parameters in S_2 are identified through equation (42). In the backward pass, the derivative of the error measure w.r.t each node output (including layer 5 of the ANFIS) toward the input end is determined, and thus the parameters of S_1 are updated by the gradient descent backpropagation algorithm.

As was discussed in Section 2.5, let the output of node i of layers j be O_i^j. In the ANFIS there are five layers: $j = \{1, 2, 3, 4, 5\}$. The node outputs depend on both the incoming signals and its parameter set in $S_1 = \{a, b, c, ...\}$. Let t represent the number of nodes for the $j-1$ layer. The node output can then be expressed as:

$$O_i^j = O_i^j(O_1^{j-1}, ...O_t^{j-1}, a, b, c, ...) \tag{45}$$

Assume the given training data set has P entries. Let E_p represent an objective function that is a function of an error measure for the p th ($1 \le p \le P$) entry of training data. This is equivalent to the difference of the target output vector T_p and the actual output vector of layer 5, O_p^5:

$$E_p = \frac{1}{2}(T_p - O_p^5)^2 \tag{46}$$

Let the error rate be represented as $\frac{\partial E_p}{\partial O_p^5}$ and calculated as:

$$\frac{\partial E_p}{\partial O_p^5} = -(T_p - O_p^5) \tag{47}$$

Let z represent the number of nodes for the j th layer. An internal node error rate can be expressed through the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^{j-1}} = \sum_{m=1}^{z} \frac{\partial E_p}{\partial O_{m,p}^{j}} \frac{\partial O_{m,p}^{j}}{\partial O_{i,p}^{j-1}} \tag{48}$$

Let α represent an antecedent parameter of the ANFIS. Let O^* represent all nodes that depend on α, then we have:

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in N} \frac{\partial E}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \tag{49}$$

where N is the set of nodes whose outputs depend on α.

The parameter α can then be updated through:

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{50}$$

where η represents the learning rate. This rate can be expressed as:

$$\eta = \frac{k}{\sqrt{\sum_{\alpha} (\frac{\partial E}{\partial \alpha})^2}} \tag{51}$$

where k represents the step size. This factor controls the speed of convergence. In the case of the ANFIS, the step size is initially set to 0.01 by default. A step size decrease rate and increase rate are set to 0.9 and 1.1 respectively by default. The step size is decreased (by multiplying it with the decrease rate) if the error measure undergoes two consecutive combinations of an increase followed by a decrease. The step size is increased (by multiplying it with the increase rate) if the error measure undergoes four consecutive decreases [19], [15].

2.7: ANFIS Overview

The ANFIS algorithm discussed in Section 2.5 and 2.6 is the basis for this thesis. This thesis makes use of the MATLAB function in the Fuzzy Logic Toolbox called ‘anfis’ [26]. There have been research papers on various ANFIS algorithms that have been configured effectively for classificiation. The paper briefly presented in [7], discussed in Section 1.5 was able to classify between two heartbeats: PVCs and normal. Configurations include decreasing the computational complexity of the layers in the ANFIS as well as increasing the number of output nodes in layer five in order to classify more than two heartbeats. However, this thesis attempts to classify heartbeats under the original ANFIS algorithm described by Jyh-Shing Roger Jang in [15].

Since the original algorithm is only able to classify between two labels, multiple ANFIS’ are run in parallel. This method is discussed in Section 4.1. There are several reasons for executing this method. The first is that multiple heartbeats can be classified effectively and compared with ANN algorithms. By

implementing several ANFIS' in parallel, an assumption is made that multiple ANFIS' in parallel is not a burden to the user in terms of computational complexity. However, it is true the ANFIS itself is computationally expensive. The second reason is that the original algorithm is preserved. This again allows for comparison with various ANN algorithms as well as ease of repeatability of the results.

The advantages of ANFIS can be first described through the advantages of Fuzzy Logic [26]. Fuzzy logic is conceptually easy to understand because the mathematical concepts described in Section 2.4 are intuitive. Fuzzy logic is flexible because each layer can add more functionality without starting the algorithm from scratch. Fuzzy logic is tolerant of imprecise data as it can model nonlinear functions of arbitrary complexity through the ANFIS. The most important advantage is that fuzzy logic is based on natural language. It is the basis for human communication because it is built on structures of qualitative description. In terms of the ANFIS itself, the advantage of the algorithm as discussed in Section 1.6 is its fast convergence speed. Smoothness is also guaranteed by interpolation. Also fuzzy sets as discussed in [37] are a depiction of prior knowledge into a set of constraints to reduce the optimization research space.

The disadvantage of ANFIS is its computational complexity. The algorithm performs slower than common classification algoirthms. There is exponential complexity with the number of rules as the number of inputs to the ANFIS increases for grid partitioning. This means fewer membership functions for each input must be used in the ANFIS to be able to classify in a fair amount of time. Lastly, is surface oscillations occur when there are an increased number of fuzzy rules. This might cause convergence issues. A solution to this issue is by implementing a checking or validation error to converge when overfitting or loss of generality occurs. ANFIS had no rule sharing. This means, rules cannot share the same output membership function [15].

2.8: Subtractive Clustering as a Preprocessor to ANFIS Classification

As discussed in Section 2.4, subtractive clustering (also called cluster estimation) is another method for partitioning input data into a FIS besides grid partition. It was proposed by Chiu in [30]. This algorithm is one of several fuzzy clustering methods: K-means clustering [31], fuzzy C-means clustering

(FCM) [32], and the mountain clustering method [33]. It has been proven that the subtractive clustering method is computationally more efficient than the mountain clustering method. FCM was proposed as an improvement to K-means clustering [16]. In the comparative study [34], it has been shown that both FCM and subtractive clustering produces similar results in fuzzy modeling with the training error of subtractive clustering considerably lower. However, a modified K-means clustering algorithm in [35] shows higher reliability in terms of both execution time and decreased error than subtractive clustering in an object recognition application. In the same paper, [35], it was shown subtractive clustering behaves better with noisy input data than K-means.

Advantages of subtractive clustering include the ability to determine the number of clusters for a given set of data under a given radial parameter discussed in this section. This is especially an advantage over FCM because the algorithm does not require a clear idea of how many clusters there should be for a given input. Another advantage is its fast execution time compared to most clustering algorithms. It has an advantage over mountain clustering because instead of considering a grid point, in the case of mountan clustering, each data point is considered. Computation in mountain clustering grows exponentially with the dimension of the problem. For example, a clustering problem with four variables and each resoltution of ten grid lines would result in 10^4 grid points that must be evaluated. Subtractive clustering is an offline clustering technique that can be used for both radial basis function networks (RBFNs) and fuzzy modeling [16].

An important difference over grid partition method in terms of preprocessing the ANFIS is that the number of membership functions does not have to be defined intitially. Subtractive clustering sets the number of membership functions based on the clusters found. The idea of fuzzy clustering is to divide the data space into fuzzy clusters, each representing one specific part of the system behavior. After projecting the clusters into the input space, the antecedent parts of the fuzzy rules can be found. The consequent parts of the rules can then be simple functions. In this way, the number of clusters equals the number of rules for the Sugeno model discussed in Section 2.5. This clustering method has the advantage of avoiding the explosion of the rule base, a problem known as the “curse of dimensionality” [16].

Disadvantages of subtractive clustering include having to set a radial parameter to determine the range of influence of each cluster. An evolutionary algorithm has been proposed in [36] and shown to perform more effectively than subtractive custering. Subtractive clustering limits membership functions for ANFIS preprocessing as gaussian. This can be a disadvantage considering other membership function types have the possibility to evaluate test data more effectively.

Unlike FCM, the subtractive clustering is non-iterative and is similar to modeling RBFNs. Clusters are considered gaussian membership functions. The program flowchart for the subtractive clustering algorithm is shown in Figure 17. Let $\{\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_n\}$ represent n input data points in an M dimensional space. The data points are first normalized within a hypercube discussed in [30]. Each data point $\mathbf{x}_i$ is considered as a potential cluster center under a density measure D_i:

$$D_i = \sum_{j=1}^{n} \exp\left(-\frac{\| \mathbf{x}_i - \mathbf{x}_j \|^2}{(r_a / 2)^2}\right) \tag{52}$$

where r_a represents a radius that defines a neighborhood or range of influence. r_a is a positive constant given by the user and ranges from zero to one. Data points outside the radius contribute slightly to the density measure. The data point $\mathbf{x}_i$ with the highest density measure D_i is considered as the center of the first cluster and is denoted as $\mathbf{x}_{c_1}$ under the density measure D_{c_1}. A new density measure is calculated in order to determine another possible cluster center:

$$D_i = D_i - D_{c_1} \exp\left(-\frac{\| \mathbf{x}_i - \mathbf{x}_{c_1} \|^2}{(r_b / 2)^2}\right) \tag{53}$$

where r_b represents the radius of the second fuzzy cluster. It has been determined in [30] that r_b is generally equal to $1.5\, r_a$. This is to prevent closely spaced cluster centers. The process of determining the cluster centers is repeated until the k th cluster center density measure D_{c_k} satisfies the condition:

$$D_{c_k} < \varepsilon\, D_{c_1} \tag{54}$$

Where ε represented as a small fraction. In [30], the factor is divided into two thresholds for accepting/rejecting cluster centers. This is because it was difficult to determine the best ε since a high ε produces too few clusters and a low ε produces too many clusters.

A membership function can then be expressed given a cluster center x_{c_k} as:

$$\mu_k = \exp\left(-\frac{\| x - x_{c_k} \|^2}{(r_a / 2)^2}\right) \tag{55}$$

where x represents an input vector. Once the cluster centers are established, the number of fuzzy rules and antecedent parameters through the membership functions are found, an optimization of the rule consequent parameters are made through the LSE in the same way discussed in Section 2.6. The method of determining consequent parameters through the subclustering algorithm is discussed in [30].

The ‘subclust’ MATLAB function from the Fuzzy Logic Toolbox is used in this thesis to generate k clusters each with cluster center x_{c_k} and standard deviation ($r_a / 2$) given the radius r_a by the user [26].

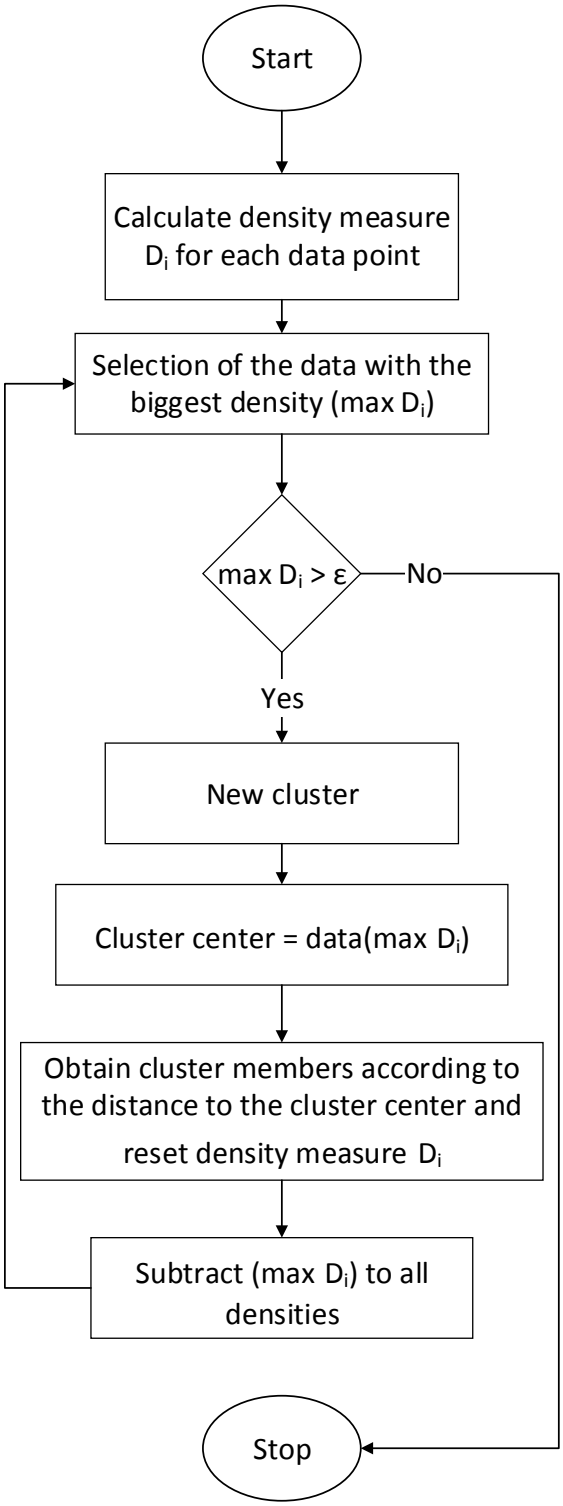


Figure 17: Program flowchart for subtractive clustering [30]

CHAPTER 3: LITERATURE REVIEW

3.1: Time Domain Based Classifier for ECG Signals

A particular algorithm that is time domain based is called the Dynamic Time Warping algorithm for classification in the paper [25]. The process can take two signals of different frame length and duration and aligns them using a nonlinear dynamic process. The Euclidean distance is taken as a decisive measurement for the difference between a reference signal and a query signal (test signal) in order to warp the signals in time so as to reduce the difference between them. A root-mean-square (RMS) error measurement then determines the similarity between the two signals. Classification was done between the normal heartbeats and arrhythmia heartbeats and the results showed a mean error of 4 milliseconds for one sample under a sampling frequency of 250 Hz.

The advantage of the Dynamic Time Warping algorithm is its simplicity, as all the features are in the time domain. Hence it is not computationally intensive to implement. Barely noticeable features such as the P, Q, S, and T waves can be matched. However, due to their small time difference, the Dynamic Time Warping algorithm must accept these differences. The disadvantage of this method of classification is the choice of reference signals. If there is a slight mismatch after time warping the query signal with respect to the reference signal, then the heartbeats will be incorrectly classified.

Dynamic Time Warping algorithm can be described as method of compressing or stretching a query signal with respective to a reference signal. Compression or stretching depends on if the query signal's has a time difference. In [25], the paper differentiates between past approaches by reducing the error in classification of a query signal after the query and reference signals have been warped together.

A RMS difference calculation to measure the amplitude difference of time aligned query and reference signals offers insight into the two signal's similarity. It is a convenient calculation because it can serve as a measure how far on average the error is from 0. The minimum normalized RMS difference, Z, means the reference signal R and query signal Q are similar and hence classified. Let R(i) and Q(i) be samples of the reference and query signals respectively. It can be calculated as follows:

$$Z = \sqrt{\frac{\sum_{j=k}^{n} [R(i) - Q(i)]^2}{\sum_{i=k}^{n} R(i)^2}} \tag{56}$$

When the RMS reaches a minimum value, the query signal is considered classified for a particular heartbeat.

3.2: Radial-Basis-Function (RBF) Neural Network with the Genetic Algorithm for ECG Classification

The RBF neural network was used as a nonlinear mapping between input and output vector spaces in [17]. The paper proposed a four stage, denoising, feature extraction, optimization and classification method for detection of PVCs. In the first stage, a wavelet denoising in noise reduction of a multi-channel high resolution ECG signal was done. A stationary wavelet transform (SWT) was used. A feature extraction module extracted ten ECG morphological features and one time interval feature. A RBF neural network with different value of spread parameter was used. The difference in past approaches with the RBF neural network is that the genetic algoirthm was used to find the best value for the RBF parameters. A classification accuracy of 100% for training dataset and 95.66% for testing dataset and an overall accuracy of detection of 95.83% were achieved over seven records (100, 101, 102, 104, 105, 106, and 107) from the MIT-BIH arrhythmia database.

For preprocessing, a stationary wavelet transform for denoising is performed through the Savitsky-Golay filter for smoothing and normalization. Using the SWT has the advantage of reducing the noise from electronic activity of muscles (EMG) and instability of electrode-skin contact. It also has the

advantage of being a time-varying transform in which the sensitivity of alignment of the signal in time is removed compared to the discrete wavelet transform (DWT). The advantage of using the Savitsky-Golay filter is it's possible to achieve high level of smoothing without attenuation of the data features. It can preserve high frequency components.

Morphological and timing features were extracted from the heartbeats. Features that describe the basic shape were the amplitudes of the P-peak, Q-valley, R-peak, S-valley, and T-peak. Features that describe the position of the waves in the window are positions of the following: P-peak, Q-valley, R-peak, S-valley, and T-peak. An RR interval ratio reflecting the deviation from a constant heartbeat rate was also extracted.

The exact RBF network was used. The number of RBF centers was made equal to the number of input vectors. Three classifications were implemented: normal, PVC, and other arrhythmia heartbeats. Therefore three neurons were used in the output layer. The output (target) vector is defined as one-hot-encoded. A random slection of 200 heartbeats from each class was made. This means 600 heartbeats were used for training. The disadvantage of using the RBF network was that the spread parameter was not optimum. The genetic algorithm was used to find the optimal spread parameter.

The advantage of the RBF network is its exectution time is fast compared to other feedforward algorithms. Multi-dimensional, non-differentiable, non-continouous, and non-parametrical problems can be solved. The disadvantage is the possibility that it will not completely classify the training data. Trial and error is a must. The number of hidden neurons for exact RBF networks is a large size when the training patters are large. The advantage of the genetic algorithm is that it is not dependent on the error surface. Limitations include the possiblily of converging towards a local minimum instead of global minimum. The genetic algorithm cannot solve certain problems due to poorly selected fitness functions.

3.3: Hyperellipsoidal Classifier for ECG Classification

One recent study on classification of ECG signals is the family of hyperbox classifiers in [8] paper. Hyperbox classifiers allow for a simple feature space structure to interpret results. Since hyperboxes do not possess differentiable boundaries, a learning process is induced. Particularly, a hybrid learning strategy of both fuzzy C-means clustering (FCM) and genetic algorithms are used. The advantage of this classification approach is that a simple structure can be used to achieve a high level of interpretability of the classification rules. The disadvantage is that the learning process is not directly suitable for gradient-based optimization techniques.

In the paper, 26 morphological features were extracted representing amplitude, area, specific interval durations, and measurements from the QRS vector in the vectorcardiographic (VCG) plane. The features were extracted from two leads from the MIT-BIH arrhythmia database.

The ECG signals from the MIT-BIH arrhythmia database were preprocessed. A notch filter was used to eliminate power-line interference. This was done through a moving average of samples in one period of the interference. A low pass filter suppressed tremor noise by the moving average of samples in a certain time interval. A high pass recursive filter was implemented for baseline suppression. An analysis was done to classify between normal and PVC heartbeats. A comparison between using a hyperbox and hyperellispsoid to classify the heartbeats was made. It was found that both structures performed reasonably well with the hyperellispoid performing slightly better. Three methods obtained a mean performance accuracy of 97.7-98.2% with the FCM combined with the genetic algorithm, 99.2-99.4% with the genetic algorithm used to provide optimal hyperboxes, and 99.5% with the genetic algorithm used to provide optimal hyperellipsoids using one or more hyperboxes/hyperellipsoids.

The FCM algorithm clusters data located in a certain feature space. Advantages include an effective method of clustering for overlapped data compared to the k-means algorithm. The k-means algorithm must have data points exclusively belonging to one cluster center. On the other hand, for FCM, the data points are assigned membership to each cluster. One disadvantage includes a priori specification of the number of clusters and Euclidean distance measurements. An objective function is applied:

$$\mathbf{Q} = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{2} \parallel x_k - v_i \parallel^2, \; x_1,...,x_n \tag{57}$$

Equation (57) represents the n-dimensional patterns defined in the space, $x_k \in R^n$, $U = [u_{ik}]$ represents the partition matrix, $\| \, . \, \|$ stands for the distance function, c is the number of clusters, and $v_1,...,v_c$ are the centers of the clusters (prototypes). Minimization is carried out for the number of clusters **c** through the partition matrix U:

$$\begin{aligned} 0 \le \sum_{k=1}^{N} u_{ik} \le N \;\; \text{for } i = 1, 2, \ldots, \mathbf{c} \\ \sum_{k=1}^{c} u_{ik} = 1 \;\; \text{for } k = 1, 2, ..., \text{N} \end{aligned} \tag{58}$$

The algorithm starts with a random distribution of clusters **c** and iteratively modifies the prototypes and the partition matrix U in order to minimize the objective function $\mathbf{Q}$. It iteratively locates the clusters in regions where there is the highest density of homogenous data. FCM is introduced in order to orient the search process of the genetic algorithm mechanism to explore only areas around the selected clusters.

The genetic algorithm optimizes the location and dimension of hyperboxes built around the **c** clusters determined by the FCM algorithm. The algorithm uses the idea of population growth through a selection of rules (parents), crossover rules (children), and mutation rules (random changes that are user-defined). It iterates until an optimal hyperbox configuration is produced.

The hyperboxes can translate into well-known format of rule-based classifications where interval limits are the respective edges of the corresponding hyperbox. There are several other geometric

constructs for classification. A hyperellipsoid, being another geometric construct, can describe all other geometric constructs through the following equation of a hyperellipsoid, where m = 4:

$$\left(\frac{x}{x_r}\right)^m + \left(\frac{y}{y_r}\right)^m + \left(\frac{z}{z_r}\right)^m = 1 \tag{59}$$

where x_r, y_r, and z_r are the equatorial and polar radii, and the parameter m determines the form or shape of the object.

3.4: Adaptive Parameter Estimation Using Sequential Bayesian Methods for ECG Classification

In [3], a ECG signal classification using estimated parameters from proposed models including adaptive multi-harmonic ECG modeling, interactive multiple modeling, and sequential Markov chain Monte Carlo modeling as features for the classifier was demonstrated in a Bayes maximum-likelihood (ML) classifier. The adaptive multi-harmonic ECG model made use of the following MIT-BIH databases: supraventricular arrhythmia, normal sinus rhythm, malignant ventricular ectopy, and atrial fibrillation. The interactive multiple model and sequential Markov chain Monte Carlo model made use of the MIT-BIH arrhythmia database.

The adaptive multi-harmonic ECG model showed 90% accuracy in classifying three of the four classes. For more information on the specific classes classified for this model and features extracted see [3]. The interactive multiple model and sequential Markov chain Monte Carlo model showed 98% accuracy in classifying the normal, LBBB, RBBB, ventricular escape, and junctional or nodal escape heartbeats.

Six features were extracted for the interactive multiple model and the sequential Markov chain Monte Carlo model. The first five features were obtained from average estimates of a coefficient from the state vector discussed in [3] at five regions in the QRS complex (local averages are used for robustness to abrupt signal changes). The sixth feature is obtained from the mean of the coefficients in the P-wave.

The classifer used in the three models is a supervised learning technique. Let $\mathbf{y}$ represent a feature vector of dimension $N_{\mathbf{y}}$. The Bayes ML classifer calculates the ranks of the likelihood $p(\mathbf{y} \mid C_q)$ of the feature vector conditioned on each of the considered classes C_q, which are assumed to follow multivariate Gaussian distributions. Considering N_q classes, the likelihood of the feature vector $\mathbf{y}$ in class C_q is given by:

$$p(\mathbf{y} \mid C_q) = \frac{1}{(2\pi)^{\frac{N_{\mathbf{y}}}{2}} \mid \sum_q \mid^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{y}-\mu_q)^T \sum_q^{-1} (\mathbf{y}-\mu_q)} \tag{60}$$

$$\text{for } q = 1, \ldots, N_q$$

where μ_q and $\sum_q$ are the Gaussian mean and covariance for class C_q. The means and covariances are determined using a set of training feature vectors from each class. For classification of a given test feature $\mathbf{y}$, the likelihood in equation (60) is evaluated for each class C_q using the corresponding mean and covariance. The classifier output is the class C^* which maximizes the log-likelihood and is expressed as:

$$C^* = \arg\max_q \log p(\mathbf{y} \mid C_q) \tag{61}$$

The N_q classes would then correspond to the various ECG signal types and the feature vector $\mathbf{y}$ is formed using the appropriate parameter estimates that are unique to each class based on the proposed ECG models in [3].

The difference between past approaches compared to this method is that there is no requirement for early-stage processing to obtain prior signal information. The reason for the success of the proposed method was its ability of features generation of Gaussian distribution of small standard deviation due to small fluctuation between features of the same arrhythmia type of different signals. The proposed models in [3] have the advantages of not requiring user-defined parameters, early stage processing to obtain *a priori* ECG signal information for filter initialization or ECG fiducial point delineation, and adaptivity to changes in ECG signal morphology. One disadvantage of the classifier is that it has a strong assumption

of feature independence. The classifier also had a small feature set and is based on a supervised learning approach, and thus does not perform well when tested with ECG heartbeats that belong to arrhythmia classes that were different from those used to train the classsifer.

3.5: Fast Fourier Transform and Levenberg-Marquardt Classifier for ECG Signals

Feature extraction was performed through the Fast Fourier Transform and the Levenberg-Marquardt algorithm to classify heartbeats in [22]. The Tachycardia, Bradycardia, Super Ventricular Tachycadria, Incomplete Bundle Branch Block, Bundle Branch Block, and Ventricular Tachycardia were classified. Four features were extracted through the Fast Fourier Transform and annotations made by cardiologists in the MIT-BIH arrhythmia database. They were the maximum QRS interval, average QRS interval, minimum QRS interval, and the heart rate. The most effective classification was done through a network of 5 hidden neurons in the first layer and 4 hidden neurons in the second layer. The accuracy of the classifer was 98.48% with a dataset of 20 heartbeats trained with another dataset of 20 heartbeats independently.

The Fast Fourier Transform was used to extract the R peaks. The Q peak and S peak were then detected through a derivative approximation in equations (3-4) of [22]. The heart rate was also used as an input feature by dividing 60 beats per minute by the RR intervals extracted by the Fast Fourier Transform.

The difference between past approaches compared to the Levenberg-Marquardt classifier is the creation of two hidden layers. The first hidden layer consisted of the hyperbolic tangent activation function. The second hidden layer consisted of the logistic sigmoid activation functions. A mean square error for ending training was set to 0.001.

The advantage of using this method is guarenteed classification of six heartbeat types discussed in [22]. This is from having two hidden layers in addition to a quick and simple approach to extracting the fundamental features for discriminating the heartbeats. One disadvantage of this algorithm is that it cannot detect other features that might improve the classifier. In addition, this network structure is not

computationally efficient because there is more than one hidden layer. Most effective networks can be built using one hidden layer.

Note the training phase of a neural network is assuming the features of an ECG signal is extracted and normalized. One advantage of the Levenberg-Marquardt Classifier is that it converges faster compared to gradient descent backpropagation algorithm discussed in Section 2.6. It combines the gradient descent and the Gauss-Newton method. The gradient descent is assured convergence through proper selection of the learning rate parameter but converges slowly. The Gauss-Newton method converges rapidly near a loca or global minimum, but also may diverge.

The algorithm involves a second order derivative matrix of the scalar-valued error functions known as the Hessian matrix. It allows the algorithm to converge faster than the learning rate parameter discussed in Section 2.6. The disadvantages of the Levenberg-Marquardt algorithm is that the calculation of the Hessian matrix is computationally expensive and needs to be positive definite. The Hessian matrix requires Jacobian matrix. It is a first order derivative matrix of the network errors with respect to the weights and biases of the neural network.

Let n represent the iteration count of the algorithm. Let $w(n)$ and $w(n+1)$ represent the weight at a specific iteration and updated weight respectively. Let $\mathbf{H}$ and $\mathbf{J}$ represent the Hessian and Jacobian matrices. Let $\mathbf{g}$ represent the gradient vector to be minimized with respect to the error vector $\mathbf{e}$. The Hessian matrix can then be calculated as an inner product of the Jacobian matrix:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} \tag{62}$$

The gradient can be computed as:

$$\mathbf{g} = \mathbf{J}^T\mathbf{e} \tag{63}$$

The update of the weights can then be expressed as:

$$\begin{aligned} w(n+1) &= w(n) - [\mathbf{H} - \lambda\mathbf{I}]^{-1}\mathbf{g} \\ w(n+1) &= w(n) - [\mathbf{J}^T\mathbf{J} - \lambda\mathbf{I}]^{-1}\mathbf{J}^T\mathbf{e} \end{aligned} \tag{64}$$

where $\mathbf{I}$ is the identity matrix and λ is a small constant. When λ is zero, the algorithm becomes the Gauss-Newton method. When large, the algorithm becomes the gradient descent with a learning rate.

3.6: Support Vector Machine (SVM) and Wavelet Decomposition for ECG Classification

A method of classification was made involving both SVM and wavelet decomposition on ECG signals in [10]. Three types of heartbeats were classified: LBBB, normal, and PVC. Wavelet decomposition was applied to obtain features for each heartbeat, namely – mean, variance, standard deviation, minimum and maximum of detail coefficients. From the MIT-BIH arrhythmia database, the accuracy of the classifier was 98.46%, 98.47%, and 99.92% for the LBBB, normal, and PVC heartbeats respectively. Four records from the database were trained and tested: records 111, 115, 116, and 119.

The advantage of SVM is its ability to define classes that are linearly separable in order to reconstruct an optimal separating hyperplane upon projecting input data into a higher dimensional space. SVM relies on kernel functions to adjust weight vectors. The disadvantage of SVM is its inability to decide an optimal kernel function in a data-dependent way [4]. Past approaches used the daubechies 4 wavelet. 25 features were extracted for each heartbeat from wavelet analysis: mean, variance, standard deviation, and minimum and maximum detail coefficients. Three RR interval features were extracted from QRS detection as proposed in [10]. The features were the RR intervals of the heartbeat, previous heartbeat, and after the heartbeat. The linear kernel function, also called the dot product, was used to train the SVMs.

A One Against One (also called One Against All) SVM was performed as discussed in [10]. The advantage of One Against One SVM is that it allows for a multiclass SVM. Three SVMs were constructed. Classification of an unknown pattern was done according to the maximum voting, where each SVM votes for one class. The disadvantage here is that multiple SVMs are necessary for multiclassification. This increases computational complexity to the proposed algorithm. One SVM classifier is trained to classify between LBBB and normal heartbeats. For this SVM, all 28 features were

used. The second SVM was trained to classify between normal and PVC heartbeats. For this SVM, only 3 RR interval features were used. The third SVM was trained to classify between LBBB and PVC heartbeats. For this SVM, again 3 RR interval features were used.

The SVM was introduced as a learning method used for binary supervised classification. It uses a training set of known objects. It can be explained as follows:

Let there be n training examples denoted as (x_i, y_i) $i = 1,...n$ where each example has d inputs where $(x_i \in \Re^d)$, and a class label with one of two values $(y_i \in [-1, 1])$. Hyperplanes, expressed by the equation: $(w)(x) + b = 0$ are used to separate the d-dimensional data perfectly into classes. w is a vector orthogonal to the hyperplane and b is a constant. A function: $f(x) = \text{sgn}((w)(x) + b)$ is used to correctly classify the training data. The geometric distance from the hyperplane to a data point is normalized by the magnitude of w:

$$d((w,b), x_i) = \frac{y_i(x_i w + b)}{\| w \|} \geq \frac{1}{\| w \|} \tag{65}$$

Eventually a hyperplane is found that maximizes the geometric distance to the closest data points. This is done through the minimization of $\| w \|$.

3.7: Past Research on ANFIS for ECG Classification

As mentioned in Section 1.5 in [7], a classification was done to detect PVC and normal heartbeats under an ANFIS classifier. The measure of actual positives (sensitivity) and negatives (specificity) identified correctly was found to be 97.92% and of 94.52% respectively. The accuracy of the classifier was 98.53%. A Sugeno ANFIS structure discussed in Section 2.5 was used on nine records of the MIT-BIH arrhythmia database. Three features were extracted: previous RR interval, the ratio between the distance RR interval following the previous one, and the QRS interval. An algorithm based on the Pan and Tompkins algorithm in [21] was used in [7] to extract the features.

In [7], five ANFIS' were created with a different structure under the trapezoidal and triangular membership functions. One of the structures used 2 membership functions for each input. Another used 3 membership functions for each input. It was shown that increasing the number of membership functions did not decrease the training and validation error significantly. The best structure was found to be 2 membership functions for the previous RR interval, 3 membership functions for the ratio between the distance RR interval following the previous one, and 3 membership functions for the QRS interval.

One advantage is the analysis of different ANFIS structures in addition to a fuzzy rule fulfillment allowed for an improved search for the most effective ANFIS structure. One disadvantage includes the choice of using grid partitioning to partition the inputs into membership functions. This approach should not be considered for a large number of inputs since the number of rules becomes too large. This adds to an increase in computational complexity.

In [37] a DWT was used to extract features in addition to five ANFIS' trained to classify normal, atrial fibrillation, PVC, ventricular fibrillation, and ventricular flutter myocardioal ischemia heartbeats. Classification accuracy was determined to be 98.24%. The ANFIS structures were trained and tested through several records of the MIT-BIH arrhythmia database, atrial arrhythmia database, and malignant ventricular arrhythmia database.

Features were extracted in [37] through a symlet wavelet. The proposed symlet wavelet based QRS detection algorithm in [37] was used to extract the slopes of five features: the Q point, R point, S point, a point between the Q and R points, and a point between the R and S points. 70% and 30% of the data were trained and tested respectively for the five ANFIS'. The gaussian bell membership function was used. It was not specified in [37] how many membership functions were used for each input.

One advantage of using the approach in [37] is that the features extracted through the DWT QRS algorithm produced a high average sensitivity. One disadvantage is the necessity of a new category of unclassified heartbeats since the ANFIS can only classifiy between two classes. This adds to a complication in verifying results with previous ANFIS classifiers.

In [38] four ANFIS classifiers: normal, congestive heart failure, ventricular tachyarrhythmia, and atrial fibrillation heartbeats were trained and tested. The databases of the MIT-BIH normal sinus rhythm, Beth Israel Deaconess Medical Center congestive heart failure, Creighton University ventricular tachyarrthymia, and MIT-BIH atrial fibrillation were used. Classification accuracy was 96.39%. Features were extracted through the Lyapunov exponents of each heartbeat. Four Lyapunov exponent features were extracted: maximum, minimum, mean, and standard deviation. The fifth layer of the ANFIS was divided into five nodes in order to generate a one-hot-encoding scheme for a binary classification.

The ANFIS classifier in [38] made use of three gaussian bell membership functions for each input. 360 training data in 850 training iterations were performed to test 360 testing data. The advantage of this method is the fact that the multiple output nodes in the fifth layer of the ANFIS allowed for a multiclassification of four heartbeat types. This eliminates the unclassified results in [37]. Lyapanov exponents have proven to be an effective descriminator of various heartbeats. The disadvantage of this method is the fact that multiple databases were used to train and test heartbeat types. This makes it difficult to construct a machine that can effectively classify multiple heartbeat types.

CHAPTER 4: CLASSIFICATION APPROACH

4.1: ANFIS of ECG Signals

Figure 18 shows an overall approach to classification. In this case, a raw signal is first preprocessed. This means the signal is filtered and annotated. Filtering involves both a low pass and high pass filter. The low pass filter filters out unwanted noise such as power noise. Power noise is around 60 Hz. The high pass filter is used to detrend or center the signal on a base-level of zero volts in order to later extract the true amplitude of the various parts of the signal. For example, the amplitude of the P wave from an ECG signal can be extracted and compared with the other P wave amplitudes.

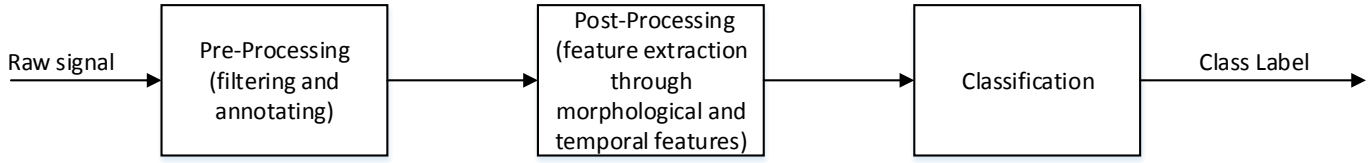


Figure 18: Process of classification

Inputs of the system consist of characteristics of the signal. Characteristics can either be annotated or calculated. Input features are composed of both temporal and amplitude characteristics of an ECG heartbeat. Let n and m represent the number of input features and the number of heartbeats to classify respectively. Obviously an increase in input features increases the system's ability to classify the beat.

Input features from an ECG signal can be extracted through a database. Figure 19 shows eight input temporal features of an ECG signal. Figure 20 shows the RR previous interval is from the R peak of the heartbeat being analyzed to the previous R peak. The RR subsequent interval is from the R peak of the heartbeat being analyzed to the subsequent R peak. Figure 21 shows the amplitudes of the P, Q, R, S, and T waves.

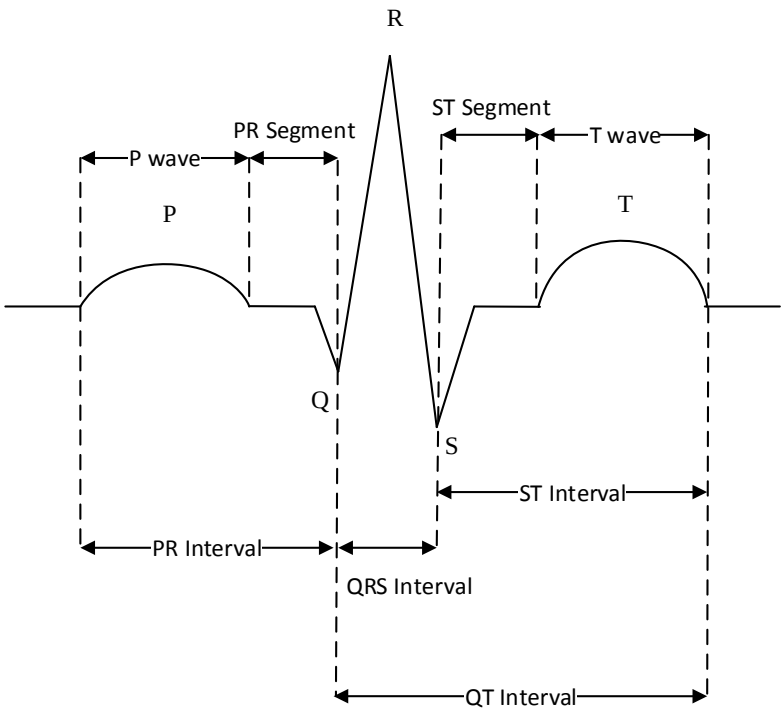


Figure 19: Several temporal input features of an ECG signal [6]

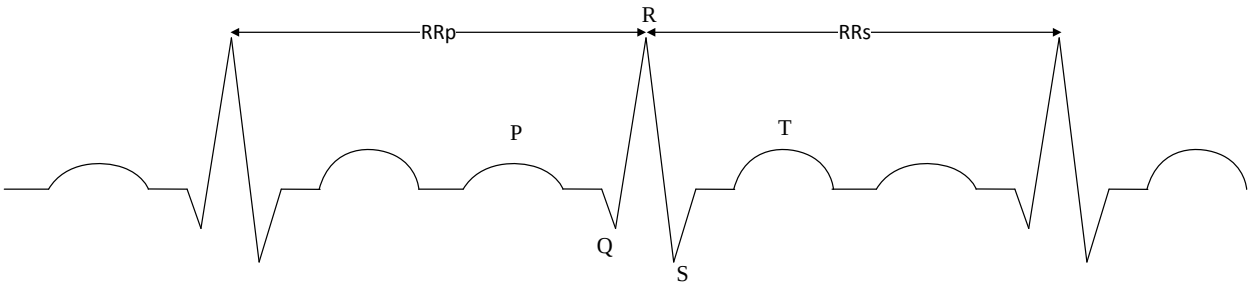


Figure 20: RR previous interval (RRp) and RR subsequent interval (RRs) input features [6]

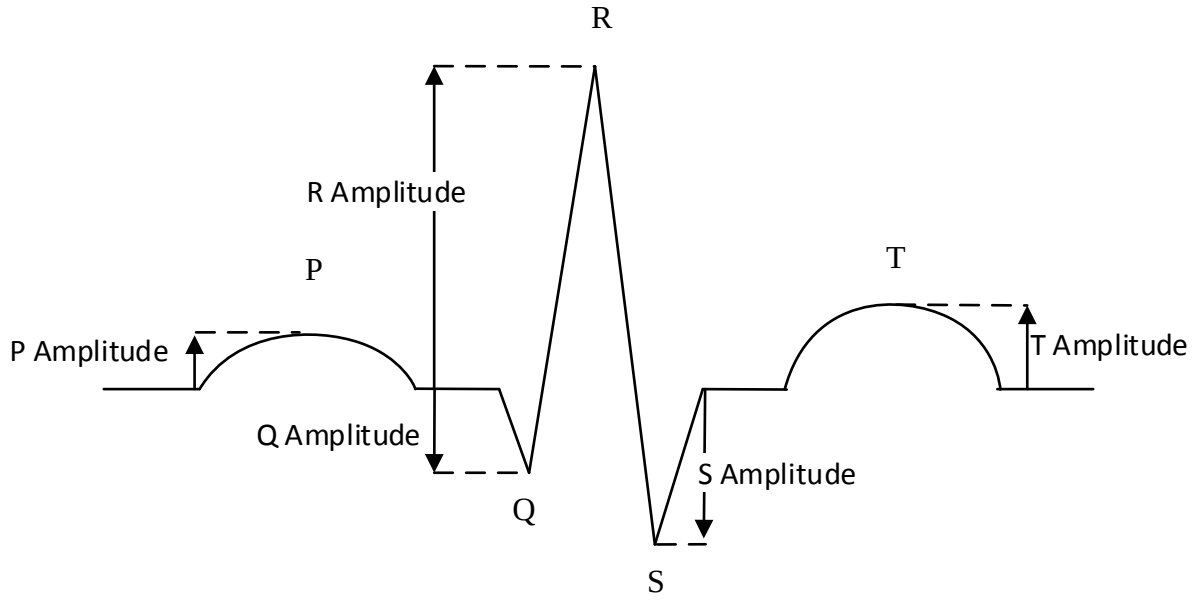


Figure 21: Amplitude input features [6]

Input features are listed in Table 2. The inputs vary based off heart rate. A normal heart rate or normal sinus rhythm is 60-100 bpm (beats per minute). The table shows seven inputs ($n = 7$) chosen based off three sources [5], [24], and [8] to be put through an ANFIS. 'NF' means that there is no feature information to discriminate the particular heartbeat. The left column shows the six heartbeats of an ECG signal to be classified. Inputs, especially the amplitudes vary between the limb leads and precordial leads. The ratio, RRs / RRp, was used as an input because it reveals variations in heart rate.

Table 2: Input features of an ECG signal [5], [8], [24]

	R Amplitude (mV)	**RRp Interval (sec)**	**RRs Interval (sec)**	**RRs/RRp**	**PR Interval (msec)**	**QRS Interval (msec)**	**ST Segment (msec)**
Normal	1.5-2	0.6-1.2	0.6-1.2	1	120-200	80-100	80-120
PVC	< 2	< 0.6	> 1.2	> 1	NF	> 120	NF
APC	> 2	< 0.6	> 1.2	> 1	NF	< 80	NF
LBBB	NF	NF	NF	NF	NF	> 120	NF
RBBB	NF	NF	NF	NF	NF	> 120	> 120
Paced	> 2	NF	NF	NF	> 280	> 120	NF

Figure 22 shows the general block diagram for classifying ECG signals for this thesis. The left side shows the seven input features used. The beats are then classified as Normal, PVC, APC, LBBB, RBBB, or Paced.

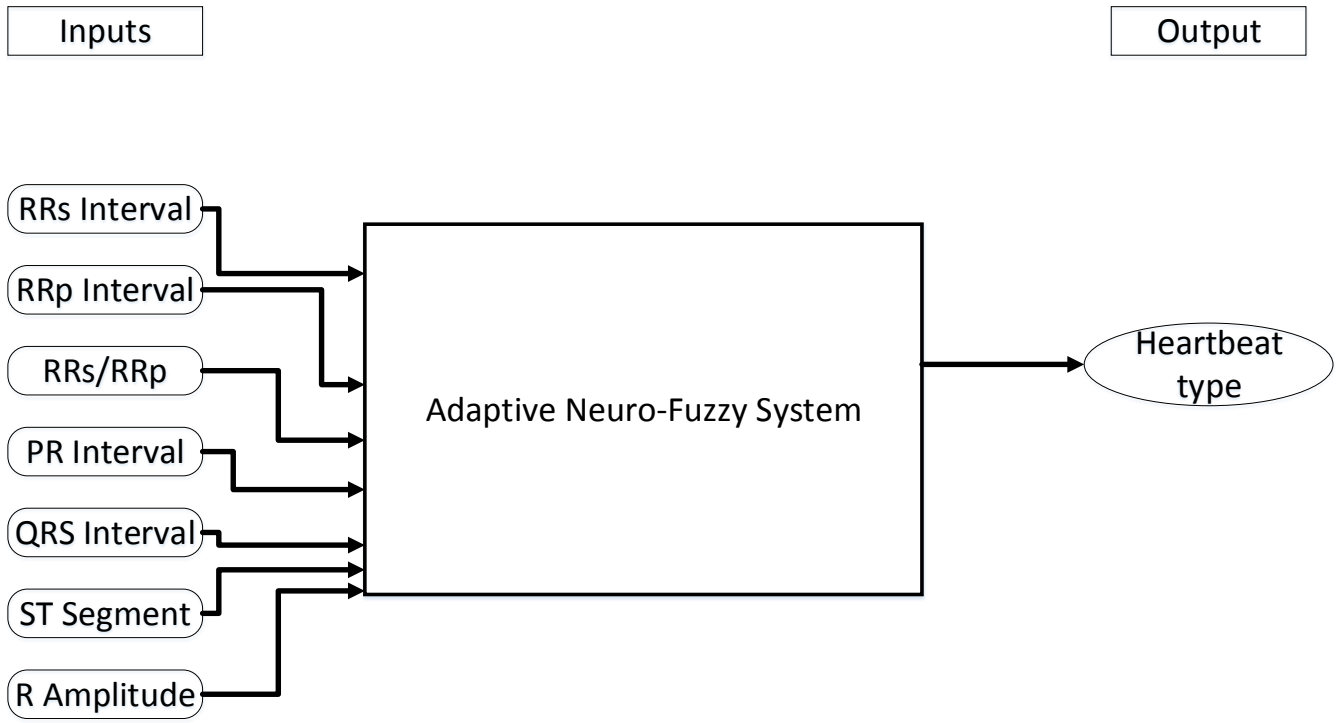


Figure 22: General ECG block diagram for this thesis

Since the ANFIS is a binary classifier, and hence one output, six ANFIS' were trained, validated, and tested. A threshold was used at the output of each ANFIS to classify a heartbeat as either a specific heartbeat (denoted as '1') or the entire set of heartbeats without the specific heartbeat being classified (denoted as '0'). Let N represent normal heartbeats, V represent PVC heartbeats, A represent APC heartbeats, L represent LBBB heartbeats, R represent RBBB heartbeats, and P represent paced heartbeats. Let All represent the combination of all six heartbeat types. Let $\overline{N}$ represent all the heartbeats without the normal heartbeats, $\overline{V}$ represent all heartbeats without PVC heartbeats, $\overline{A}$ represent all the heartbeats without the APC heartbeats, $\overline{L}$ represent all the heartbeats without the LBBB heartbeats, $\overline{R}$

represent all the heartbeats without the RBBB heartbeats, and $\overline{P}$ represent all heartbeats without the paced heartbeats. The method of classifying is illustrated in Figure 23.

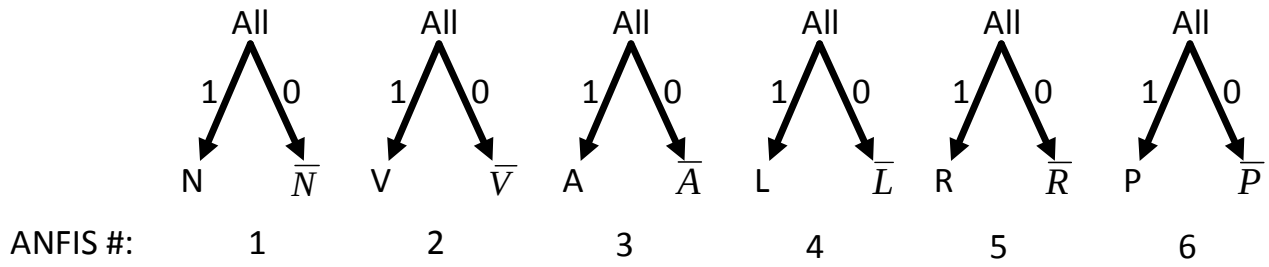


Figure 23: Method of classification with ANFIS

A threshold after each ANFIS determines if the heartbeat is the specific heartbeat denoted as a target of '1' or all heartbeats without the specific heartbeat denoted as a target of '0'. Let f represent the output of an ANFIS. Let f_{Th} represent the value of the output after a threshold, either '0' or '1'. The threshold is as follows:

$$\text{If } f < 0.5, \text{ then } f_{Th} = 0;$$
$$\text{Else } f_{Th} = 1$$

Figure 23 shows each ANFIS structure based on Figure 15. This diagram shows the internal functionality of the system. First, each input feature is passed through two membership functions for the case of grid partitioning or k clusters for the case of subtractive clustering. Two membership functions, for example, would capture the 'high' and 'low' linguistic information for each input feature. Three membership functions would capture the 'high', 'medium', and 'low' linguistic information for each input feature. For more membership functions, linguistic information would encompass more ranges for each input feature. Since the number of input features is seven, the number of nodes for each layer can be calculated. For the first layer, the number of nodes was calculated to be 14 nodes (grid partition) and $k \cdot n$

nodes (subtractive clustering). For the second, third, and fourth layers, the number of nodes was calculated to be 128 nodes (grid partition) and k nodes (subtractive clustering).

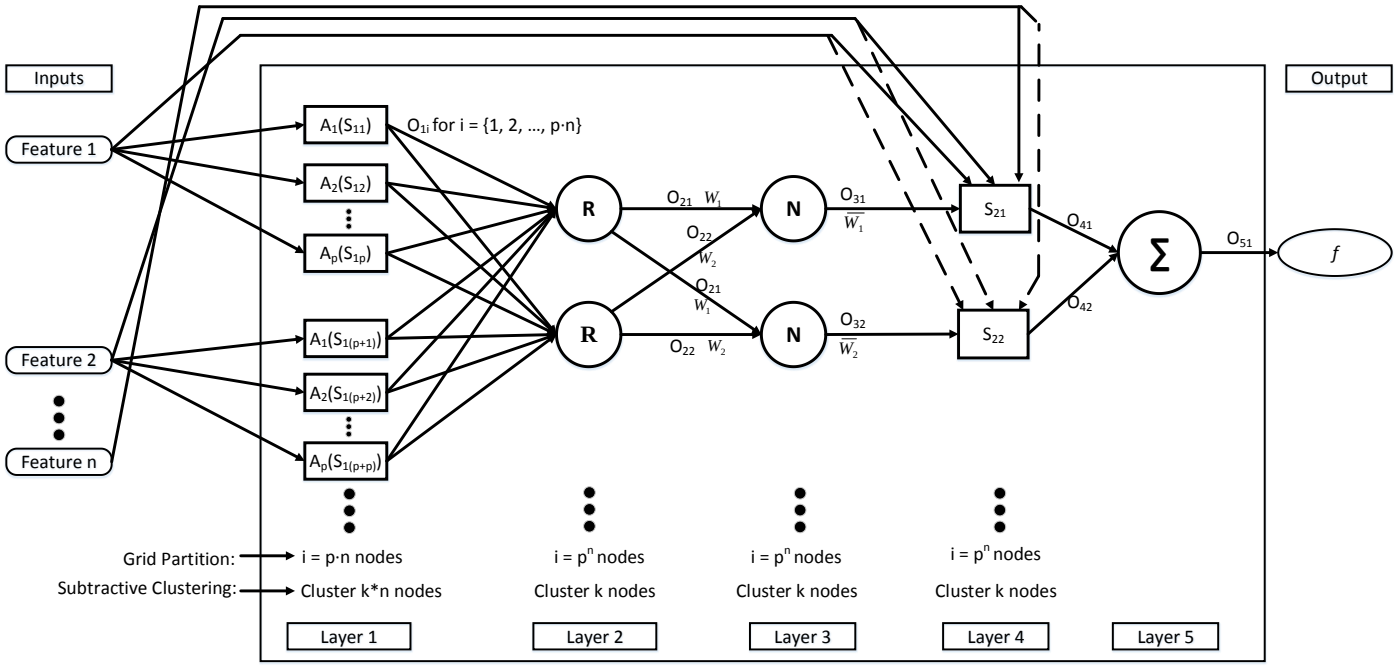


Figure 24: General ANFIS structure

4.2: Extracting Input Features

It is necessary to annotate the signal to extract the features. The MIT-BIH website, collectively known as Physionet, holds multiple tools for analyzing and annotating various heart signals (most being ECG records). This ECG classification is based off the Arrhythmia Database [11].

In order to begin preprocessing the ECG records of the database, the signal files had to be downloaded from Physionet's 'Physiobank Automated Teller Machine', this machine allows for a graph of each signal from 10 second-length signals to the full 30 minute-length signals. The full 30 minute-length signals were chosen for a complete analysis of each patient's record. For the purpose of analyzing the signal in MATLAB, data ('.dat'), header ('.hea'), and annotation ('.atr') files for each record needed to be downloaded. The data file holds the amplitude values and samples for two leads. Most records consist of a modified limb lead II signal, also called MLII, and a "lower" precordial lead such as V1

(occasionally V2 or V5, and in one instance V4). For example, the first record: record '100', consists of both the MLII and V5 leads [11].

A MATLAB toolbox, called the Physionet Waveform Database (WFDB) toolbox, is specialized for analyzing the ECG signals. It can be downloaded from the Physionet website. This toolbox consists of functions that annotate the signal and specialized algorithms for detecting various points of an ECG signal. One of the annotation functions allow for classifying each heartbeat as normal or abnormal as determined by two cardiologists. Several records could not be simulated under the toolbox. Table 3 shows the MIT-BIH records and the number of corresponding heartbeat types. Another function that uses an algorithm for detecting the QRS complex onset and offset of an ECG signal is called the Pan-Tompkins algorithm. The same function detects the P wave onset and offset as well as the T wave onset and offset [11]. Two cardiologists detected the P wave while the T wave was detected through the Pan-Tompkins algorithm

Preprocessing MATLAB functions, some from the WFDB toolbox and some made from scratch are defined below (a more comprehensible view of these functions can be found in Appendix B):

rdsamp: Reads the ECG records from the MIT-BIH arrhythmia database. The record name is taken as input as well as the number samples to read from. The amplitude of the signal is then outputted as well as the sampling rate.

l_or_hpf: Takes the one dimension ECG signal, desired cutoff frequency, butterworth order, and a choice between performing a low or high pass filter as input arguments. In this thesis, a 30 Hz cutoff, 3^{rd} order butterworth low pass filter and a 1 Hz cutoff, 3^{rd} order butterworth high pass filter is used. The design of these filters was taken from [8].

rdann: Takes the annotation file of the specific record to extract the R peaks and heartbeat types for each heartbeat.

sqrs: Extracts the QRS points under Pan-Tompkins algorithm of each heartbeat given the annotation file.

ecgpuwave: Extracts the P and T peaks as well as the onset and offsets of each point.

input_layer: Takes the samples of each point from the ECG signal as well as the signal itself. The output is the seven input features extracted as discussed in Table 2.

hist: Generates a histogram of each feature extracted for verification that input features were extracted correctly.

Table 3: MIT-BIH records and corresponding heartbeat types: 'N' being normal, 'V' being PVC, 'A' being APC, 'L' being LBBB, 'R' being RBBB, and 'P' being paced beats

Record	N	V	A	L	R	P
100	2239	1	33	0	0	0
101	1860	0	3	0	0	0
102	99	4	0	0	0	2028
103	2082	0	2	0	0	0
104	163	2	0	0	0	1380
105	2526	41	0	0	0	0
106	1507	520	0	0	0	0
107	0	9	0	0	0	2078
108	1739	17	4	0	0	0
109	0	38	0	2492	0	0
111	0	1	0	2123	0	0
112	2537	0	2	0	0	0
114	1820	43	10	0	0	0
115	1953	0	0	0	0	0
116	2302	109	1	0	0	0
117	1534	0	1	0	0	0
118	0	16	96	0	2166	0
121	1861	1	1	0	0	0
122	2476	0	0	0	0	0
123	1515	3	0	0	0	0
200	1743	826	30	0	0	0
201	1625	198	30	0	0	0
202	2061	19	36	0	0	0
203	2529	444	0	0	0	0
205	2571	71	3	0	0	0
208	1586	992	0	0	0	0
209	2621	1	383	0	0	0
212	923	0	0	0	1825	0
213	2641	220	25	0	0	0
214	0	256	0	2003	0	0
215	3195	164	3	0	0	0
220	1954	0	94	0	0	0
222	2062	0	208	0	0	0
223	2029	473	72	0	0	0
228	1688	362	3	0	0	0
230	2255	1	0	0	0	0
231	314	2	1	0	1254	0
233	2230	831	7	0	0	0
234	2700	3	0	0	0	0

The Pan-Tompkins algorithm, discussed in [21], detects the QRS complexes and T intervals based upon digital analyses of slope, amplitude, and width. Thresholds are adjusted periodically to adapt to QRS morphology changes and heart rate throughout the signal. In evaluating the MIT-BIH arrhythmia database, the algorithm failed to properly detect only 0.675 percent of the heartbeats. The disadvantage of this method is the threshold technique is only useful if the heart rate is regular. For the case of irregular heart rates, two thresholds are reduced by half in order to increase the sensitivity of detection in order to avoid missing valid heartbeats.

Figure 25 shows the program flowchart of the Pan-Tompkins algorithm for detecting QRS complexes. For all transfer functions used in this algorithm, please refer to [21]. First, a DC drift and normalization is performed. The signal is subtracted by the overall mean of the signal and then normalized at each sample. This straightens out the signal along the zero volt base-line. A digital bandpass filter consisting of a cascade of a low pass filter and a high pass filter with a passband from 5-15 Hz is then applied to the signal. This reduces the influence of muscle noise, 60 Hz interference, base-line wander, and T-wave interference. A derivative filter is then applied to provide the QRS complex slope information. A squaring function is applied to make all data points positive and does nonlinear amplification of the output of the derivative emphasizing the higher frequencies (i.e. predominantly the ECG frequencies). The squaring is done by squaring the signal point by point. A moving-window integration is performed to obtain waveform feature information in addition to the slope of the R wave. Let T represent the sampling period. Let N represent the number of samples in the width of the integration window. For our case of 360 samples per second, the window is 30 samples wide (150 ms). The difference equation of this integration is:

$$y(nT) = (1/N)\,[x(nT-(N-1)T) + x(nT-(N-2)T) + ... + x(nT)] \tag{66}$$

A lower set of thresholds is then placed based on the bandpass filtered ECG signal. An upper set of thresholds is placed based on the moving-window integration. The higher of the two thresholds in each of the two sets is used for the first analysis of the signal. The lower threshold is used if no QRS is

detected in a certain time interval so that a seach-back technique is necessary to look back in time for the QRS complex. Thresholds of a signal and noise peak detect the QRS complex. They are expressed in equations (12-20) referenced in the paper, [21].

Two RR intervals averages are calculated and maintained in [21]. The reason for maintaining the RR intervals is to be able to adapt to quickly changing or irregular heart rates. When an RR interal is less than 360 ms, a judgement is made to determine whether a QRS complex or a T wave is identified. If the maximal slope that occurs during this waveform is less than half of the QRS waveform that preceded it, it is identified to be a T wave; otherwise, it is a QRS complex.

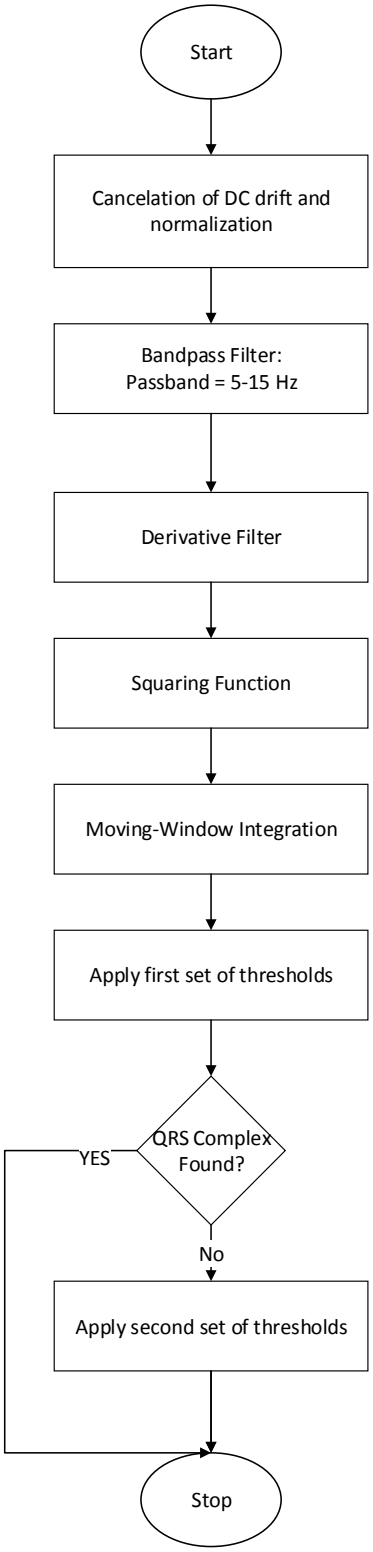


Figure 25: The Pan-Tompkins algorithm for QRS complexes [21]

Figure 26 shows the result of the function that annotates and detects the various points of MIT-BIH arrhythmia database record '100' modified limb-lead (MLII) signal. The full 30-minute length and 360 samples per second ECG signal consist of about 646,400 samples. Normal heartbeats are denoted as 'N' in the figure. The P wave, QRS complex, and T wave has been detected along with an onset and offset.

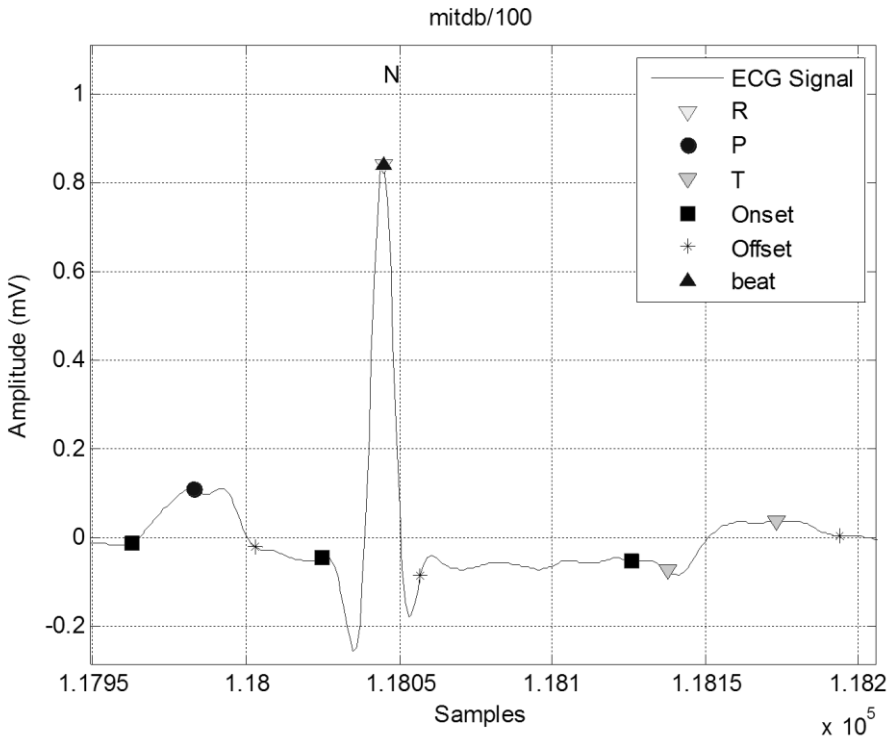


Figure 26: One heartbeat showing the results of the WFDB toolbox function

One problem with the function was the occasional detection of two T wave points as shown in Figure 26. Most signals plot one T wave point. For this reason, in addition to a lack of sources to discriminate the T amplitude feature, the feature was not used in the ANFIS classifier.

Histograms of these three heartbeats were plotted for each input to verify that the extraction of the features in Table 2 from the annotation function was done correctly. Figure 27 shows a histogram of normal P amplitudes.

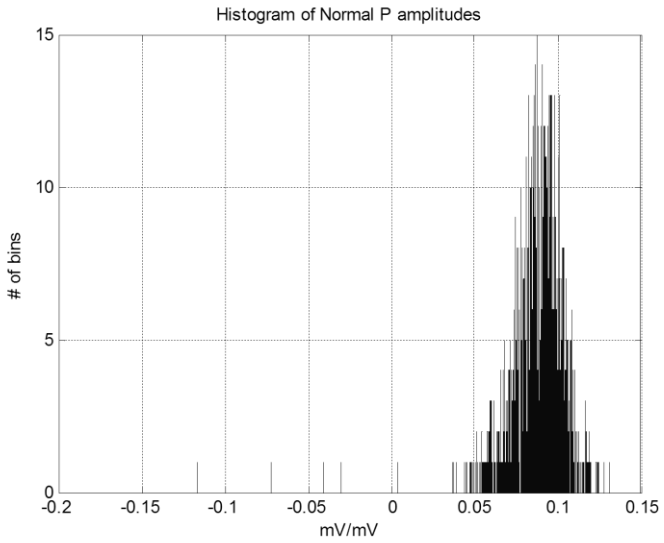


Figure 27: Histogram of normal P amplitudes

4.3: Program Flow

Figure 28 shows a program flowchart for classification of the ECG record(s). The program starts with choosing which ECG record(s) to classify. Normalization of the input features from the ECG signals was done through the subtractive clustering algorithm. The signal was detrended and cleansed of noise through a high and low pass filter for preprocessing. Seven features are extracted as discussed in Table 2. Since the ANFIS has one output, the output vector is created. Each output index of the vector is assigned to a specific heartbeat type. For example, normal heartbeats would be defined as '1' and other heartbeat types (five other heartbeats) would be defined as '0'. An input data matrix of features was generated and concatenated with an output vector. The matrix was divided for each heartbeat type into training data (55% of the input data), checking data for validation (10% of the input data), and testing data (35% of the input data). 100 random heartbeats were selected from six specific heartbeat records that were randomly selected from the database. Therefore only specific heartbeat records with more than 100 heartbeats of the specific type were selected.

Classification begins by choosing the number and type of membership functions for the ANFIS. Figure 13 and equations (6-9) show the membership functions to choose from. It is good practice to start

with a triangle membership function [15]. If computational performance is an issue, a gaussian membership function would be used due to having the smallest number of antecedent parameters. If smoothness is required to perform a better classification, then a gaussian-bell would be an effective membership function for fuzzification. Fuzzification is performed by generating an initial FIS. This could either be done by grid partitioning or subtractive clustering.

The number of membership functions needs to be taken into account with respect to the generation of fuzzy rules as discussed in Section 2.5. The MATLAB function 'anfis' can only support at most 256 fuzzy rules [11]. Since there are seven inputs and the number of fuzzy rules is equivalent to number of membership functions p to the power of the number of inputs n for grid partitioning, three membership functions is not supported. For this reason, two membership functions are chosen for grid portioning: representing linguistically 'high' and 'low' partitions of each input feature.

The number of antecedent and consequent parameters can then be calculated using equation (10) and (15) respectively for grid partitioning. They were calculated as 42 and 1024 respectively.

The rule layer as discussed in Section 2.5 was chosen to be an AND function. This means the output of the rule layer is a minimum of two membership functions as opposed to the maximum through the OR function.

The ANFIS algorithm generates an output FIS for training, validating, and testing data. Training and checking (also called validation) RMSE (root-mean-square error) is calculated for each data sample i. The RMSE compares the desired output value y, and the actual output of the FIS $\hat{y}$. Let t represent the number of heartbeats of training. The RMSE for training can be expressed as:

$$RMSE = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(y_i - \hat{y}_i)^2} \tag{67}$$

The RMSE gives a more accurate value of the error between a model (output of FIS) and observed data (training/validation data output value). There are statistical properties such as variance and

standard deviation that makes RMSE a desirable measurement. It is desirable to have a RMSE decrease or converge as the number of iterations increase.

The RMSE of the checking data is used to prevent overfitting. Overfitting occurs when the RMSE of the checking data increases. It is a result of fitting the fuzzy system to the training data so well that it no longer fits the testing data effectively. This leads to a loss of generality. The ANFIS algorithm chooses model parameters associated with the minimum checking error prior to overfitting (if it exists). Once RMSE of the training data is shown to decrease as the number of iterations increases and overfitting is eliminated, evaluation of the ANFIS is made.

In evaluating the ANFIS, the trained FIS is used to evaluate the test data. The output is a result of classification. A threshold, as was discussed in Section 4.2, on the training, checking, and testing output vectors is used to convert fractional numerical assignments for each heartbeat to integers, either ‘0’ or ‘1’.

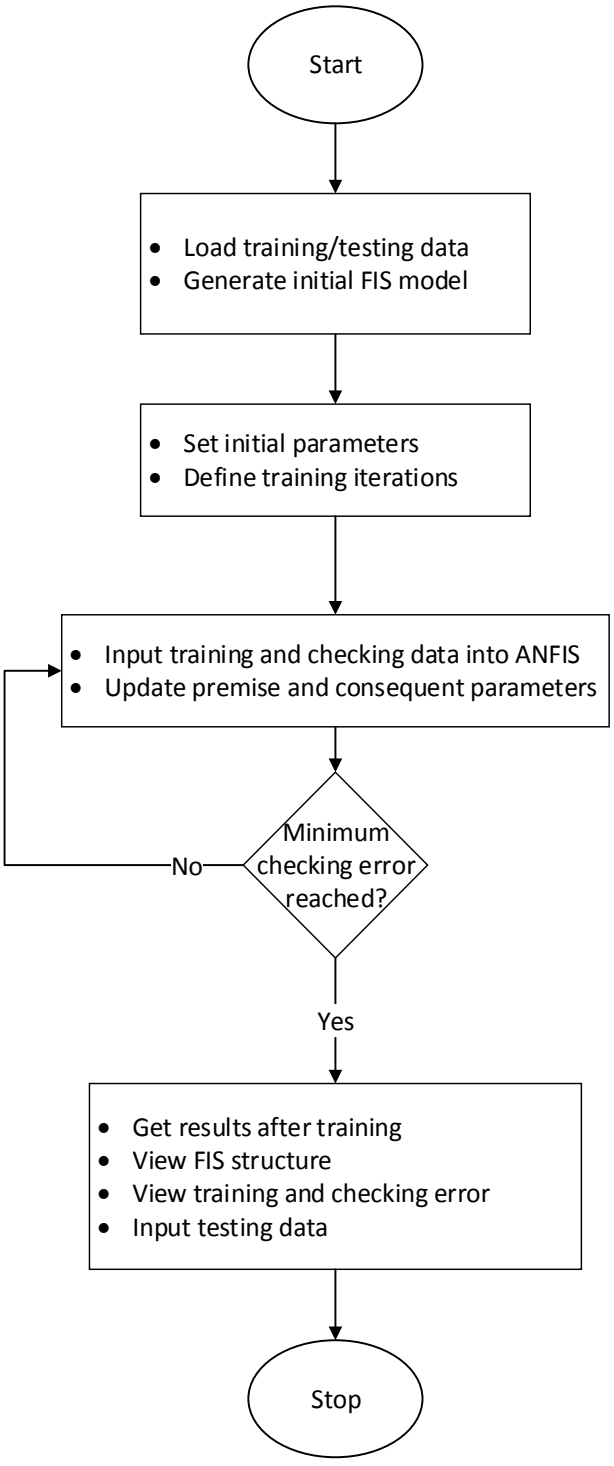


Figure 28: Classification program flowchart

4.4: Performance Evaluation Method

As discussed in Section 3.7: accuracy, sensitivity, and specificity can be used as performance measurements to evaluate the effectiveness of a classifier. All three measurements include heartbeats that define true positive, true negative, false positive, and false negative. Let TP represent the true positive heartbeats classified. Let TN represent the true negative heartbeats classified. Let FP represent the false positive heartbeats classified. Let FN represent the false negative heartbeats classified. Let N represent the total number of heartbeats being classified. The three performance measurements can then be expressed as [8]:

$$Accuracy\ (\%) = \frac{TP + TN}{N}\ x\,100\% \tag{68}$$

$$Sensitivity\ (\%) = \frac{TP}{TP + FN}\ x\,100\% \tag{69}$$

$$Specificity\ (\%) = \frac{TN}{TN + FP}\ x\,100\% \tag{70}$$

CHAPTER 5: SIMULATION RESULTS

5.1: ANFIS on Several ECG Records under Subtractive Clustering

For ANFIS classification results under grid partitioning see Section 5.4. Before implementing the program flowchart of Figure 28, several ECG records from the MIT-BIH arrhythmia database are chosen. The seven input features as shown in Table 2 are extracted using both the Pan-Tompkins algorithm and the annotations made by the cardiologists. Table 4 shows the input features by input number. This table is important for discussing the FIS membership functions illustrated later in this section. Since the ANFIS classifies between only two classes, six ANFIS' were created for classification method based on Figure 23. The heartbeat types are normal, PVC, APC, LBBB, RBBB, and paced.

Table 4: The seven features associated with a corresponding input number

Input	1	2	3	4	5	6	7
Feature	QRS Interval	PR Interval	R amplitude	ST Segment	RRs/RRp	RRp Interval	RRs Interval

The records chosen (shown in Table 5) to train, check (validate), and test were the second modified limb leads (MLII) randomly selected for each specific heartbeat. The heartbeat types extracted from the records were normal, PVC, APC, LBBB, RBBB, and paced. Each heartbeat type was divided into 55% training, 10% checking, and 35% testing data as discussed in Section 4.3. A justification for the record choice is discussed in Section 5.2. Three trials of training, validating, and testing six ANFIS' were done. This allowed for a more comprehensible classification across the entire database. In training and validating the six ANFIS' it was found that using three membership functions for each of the seven input features produced a reasonable balance between interpretability through the rules and effective classification results.

An analysis of the parameters updated through the learning rate can be performed through the training and checking (validation) error curves. This gives a general observation of the parameters being updated. Since the learning rate of equation (51) is dependent on the initial step size, two trials were

analyzed to obtain the best convergence with the least amount of oscillations: the first being when the initial step size is 0.01 by default, the second being when the initial step size is low (low learning rate).

Table 5: Heartbeats (including each heartbeat type) chosen from records of the MIT-BIH database randomly

Total	Training	Checking	Testing
600	330	60	210

Starting with the first ANFIS from Figure 23, a classification was made between normal heartbeats N and all six heartbeat types without the specific normal heartbeats denoted as $\overline{N}$. By setting the radius r_a to 0.45 for subtractive clustering as discussed in Section 2.8, three clusters (gaussian membership functions) were generated for each input. As discussed in Section 2.8, the subtractive clustering algorithm normalizes the input features about a hypercube. Normalization is also performed in the ANFIS' at layer 3 for the weights. The initial FIS generated from subtractive clustering was then fed into the ANFIS for 1000 iterations. Figure 29 shows the RMSE curves for the training and checking (validation) data under the default learning rate. It can be observed that overfitting of the data starts around 80 iterations.

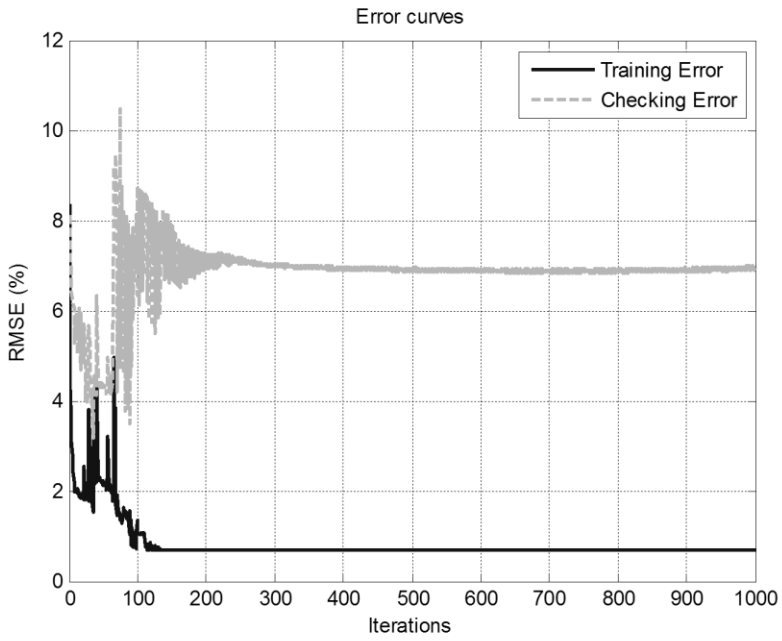


Figure 29: RMSE training and checking curves for the first ANFIS under three membership functions for each input for 1000 iterations. Initial step size is 0.01.

Since there were oscillations in Figure 29, the learning rate was decreased by setting the initial step size to 0.001. Figure 30 shows the result of decreasing the learning rate. Convergence is usually reached at minimum checking error. However, the checking error in this ANFIS did not increase as overfitting was not apparent. Overfitting would occur when the checking error starts to increase. Convergence was then chosen to be 78 iterations where the training error begins to flatten out. The training RMSE dropped from 8.38% to 1.82%.

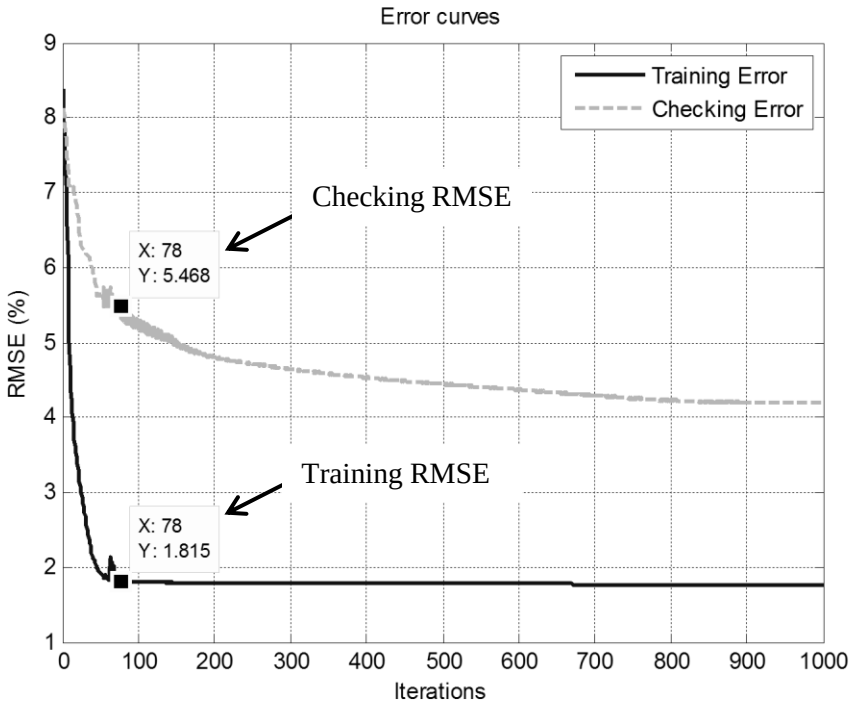


Figure 30: RMSE training and checking curves for the first ANFIS under three membership functions for each input for 1000 iterations. Initial step size is 0.001. Convergence was reached at 78 iterations.

Figure 31 shows the step curve recording the step size during training of the first ANFIS. This step size profile serves as a reference for adjusting the initial step size and corresponding step size increase and decrease rates. Usually, the step size profile is a curve that increases initially, reaches some maximum, and then decreases for the remainder of the training.

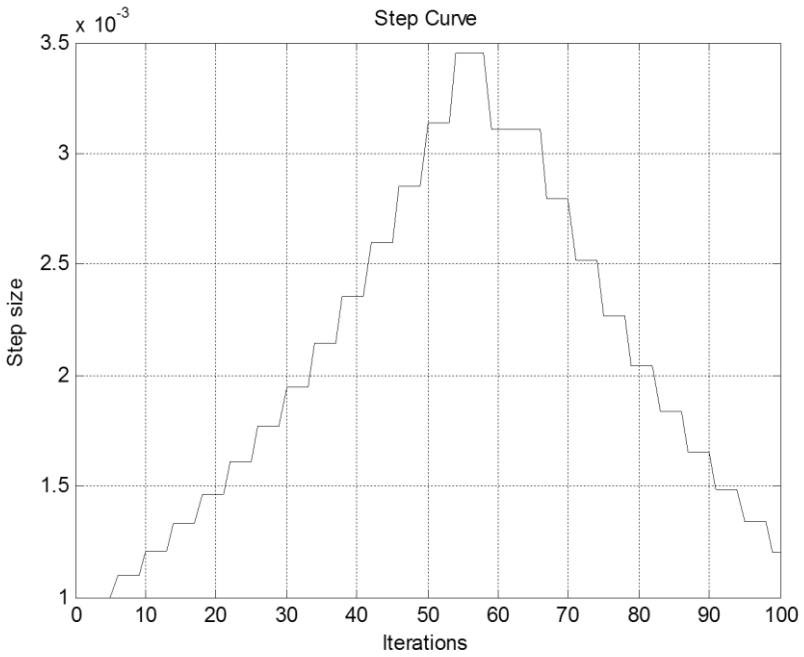


Figure 31: Step curve for the first ANFIS under three membership functions for each input for 100 iterations. Initial step size is 0.001.

Figure 32 shows the initial FIS generated from subtractive clustering. For all membership functions, the input range is determined by the lowest and highest value input. The antecedent parameters (standard deviation and center gaussian function parameters) are then fed into the ANFIS for adaptation. Figure 33 shows the adapted FIS. It is obvious that the first input membership functions show the most adaptation. This means the QRS interval (first input) is an exceptional feature for training the ANFIS.

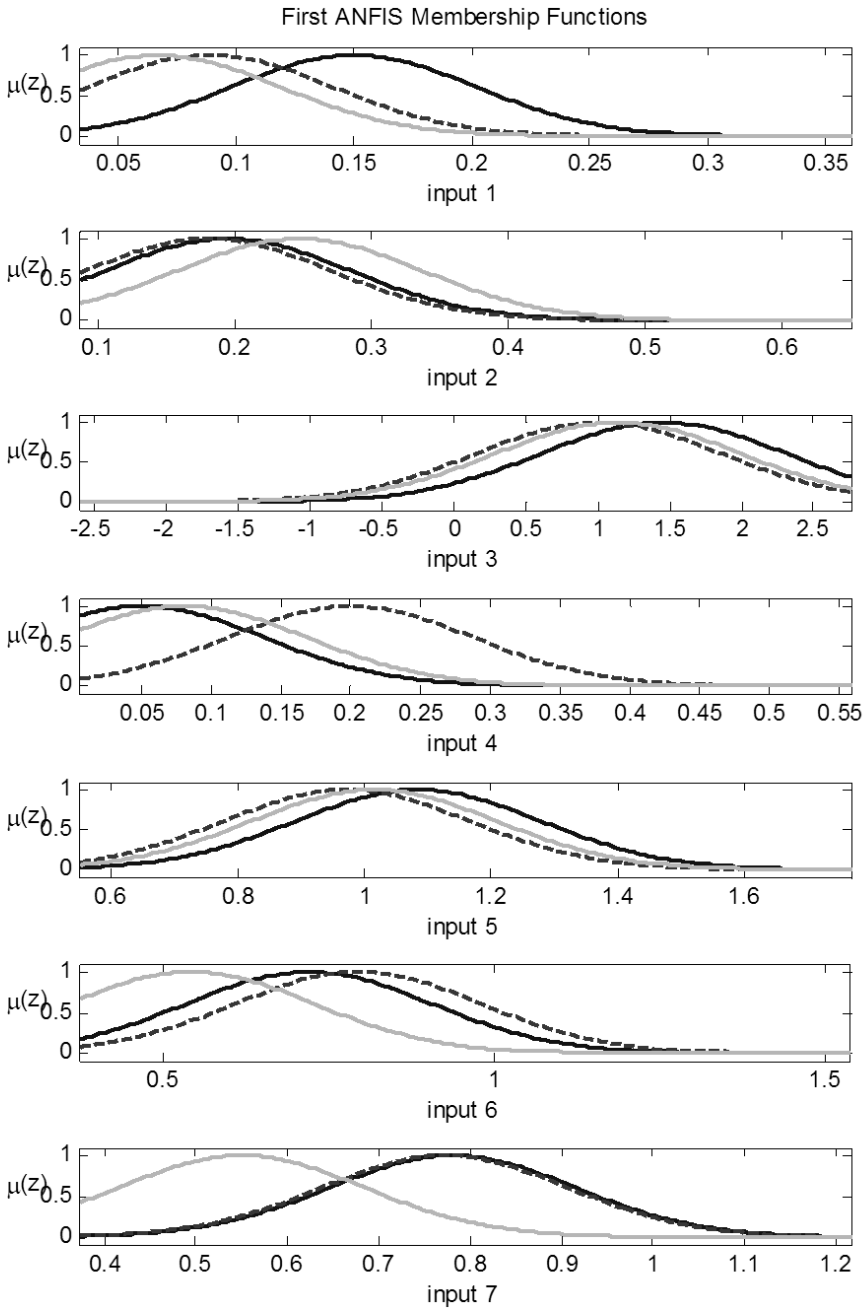


Figure 32: Initial subtractive clustering FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function. The gray function represents the third membership function.

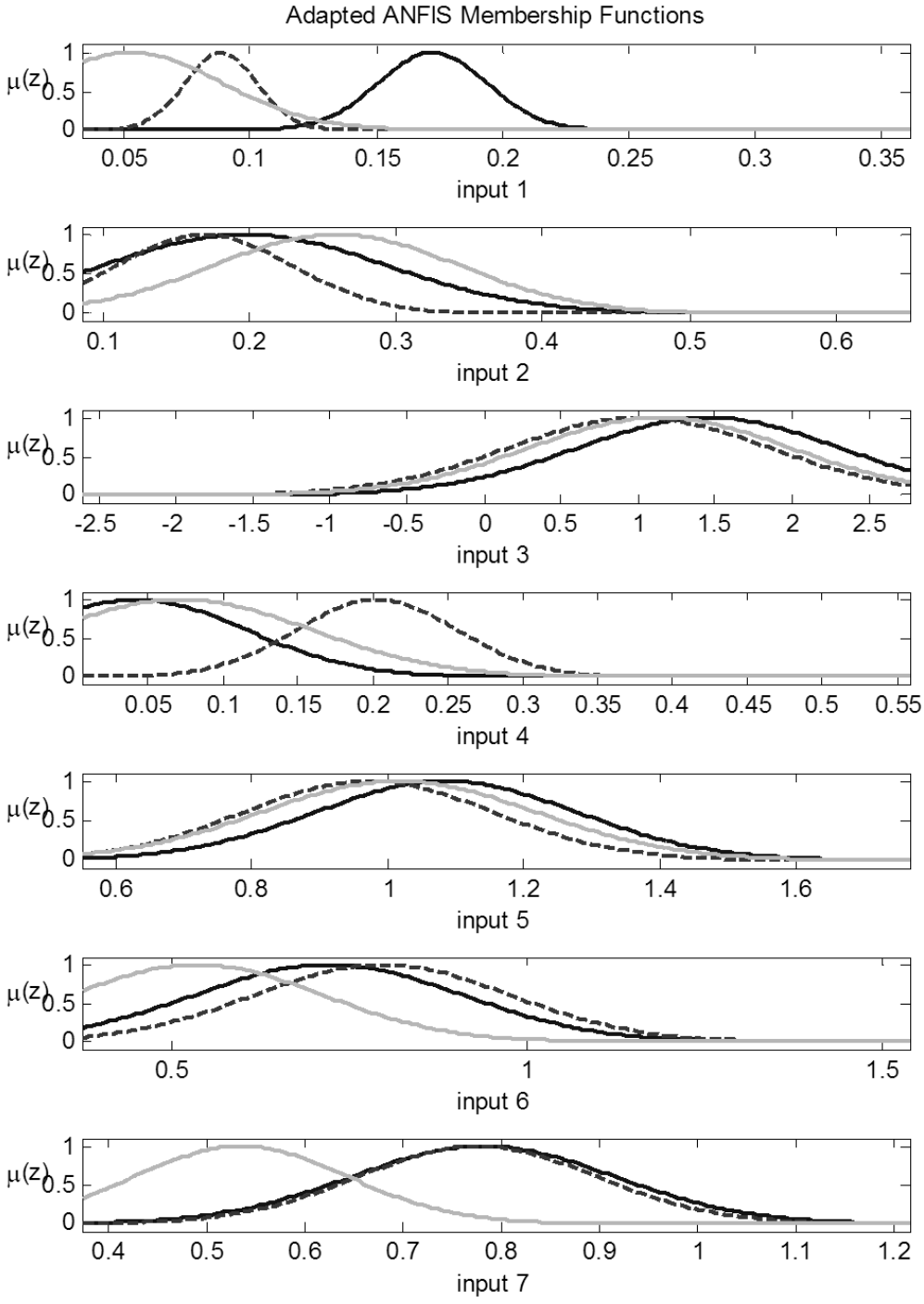


Figure 33: Adapted FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function. The gray function represents the third membership function.

Figure 34 shows an example of the first input's first membership function parameters generated from subtractive clustering and adapted after 78 iterations from the first ANFIS. As discussed in Section 2.5, a and b represents the center and standard deviation respectively for the gaussian membership functions. Figure 34 also shows the first rule's consequent parameters generated from subtractive clustering and adapted after 78 iterations from the first ANFIS.

Antecedent Parameters:

$$\{a, b\} = \{0.15, 0.052\}$$

$$\downarrow$$

$$\{a, b\} = \{\ 0.17, 0.021\}$$

Consequent Parameters:

$$\{q_{11}, q_{21}, q_{31}, q_{41}, q_{51}, q_{61}, q_{71}, r_1\} = \{1.06, 0.031, 0.029, -0.019, 0.53, 0.50, -0.62, -0.74\}$$

$$\downarrow$$

$$\{q_{11}, q_{21}, q_{31}, q_{41}, q_{51}, q_{61}, q_{71}, r_1\} = \{-0.00022, 0.00016, -0.000023, 0.00079, 0.0010, 0.00074, 0.0013, -0.00061\}$$

Figure 34: Antecedent and consequent parameters before and after training for subtractive clustering FIS. This is the first input's first membership function parameters for the antecedent parameters and the first rule's consequent parameters.

Figure 35 shows a decision surface generated from the first ANFIS. This surface is generated between the first input (QRS interval) and the fifth input (RR ratio). There are 21 different surfaces that can be observed because there are seven inputs. This is one of the advantages of ANFIS because a user can interpret this surface as to how the inputs were mapped based on the ANFIS.

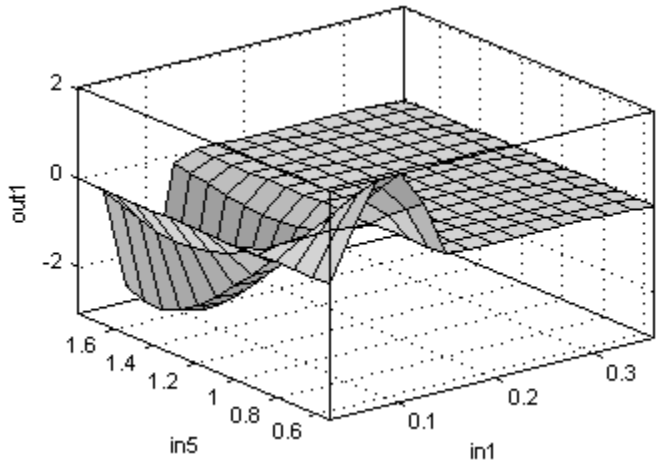
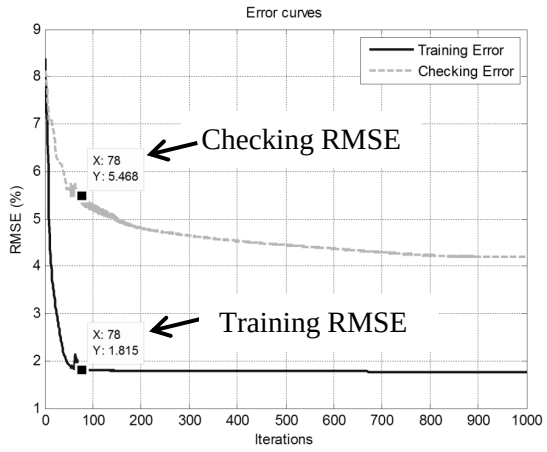


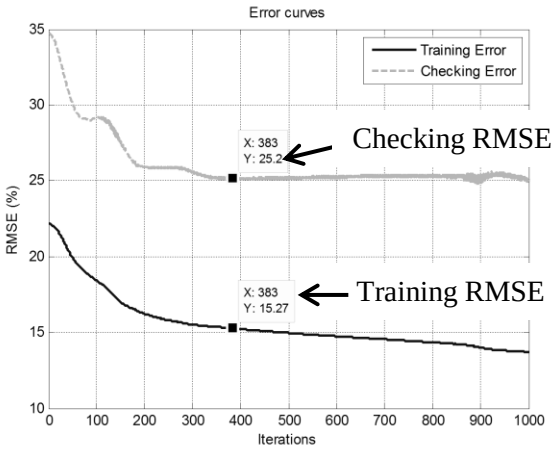
Figure 35: Decision surface generated from the first ANFIS after 246 iterations between the first input (QRS interval) and the fifth input (RR ratio)

Testing data was then evaluated through the ANFIS. As discussed in Table 5, 210 heartbeats were evaluated: 35 for the specific heartbeat (N for the first ANFIS) and 175 for the heartbeats without the specific heartbeats ($\overline{N}$). Evaluation of the first ANFIS showed all heartbeats correctly classified. This means there was 100% accuracy, sensitivity, and specificity.

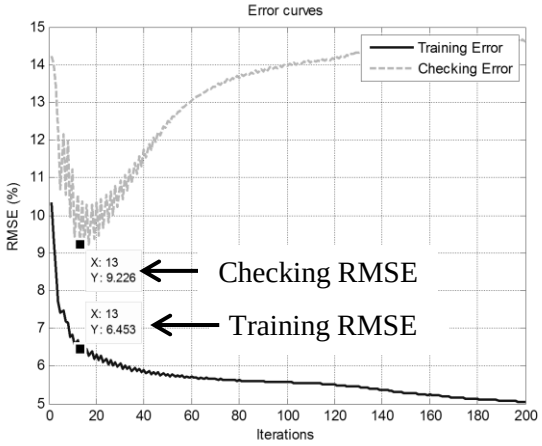
The rest of the five ANFIS' were trained, validated, and tested under the same procedure. The procedure was to develop the best trained ANFIS in order to obtain an effective classification. The best trained ANFIS is through trial and error by specifying the radius r_a for subtractive clustering in order to obtain three membership functions for each input, and the initial step size for controlling the learning rate. Convergence is reached when either the checking RMSE starts to overfit (increase). In other words the iteration with the minimum checking RMSE defines convergence, or when the training RMSE itself levels off while checking RMSE continues to decrease or levels off. Figure 36 shows the best training and checking RMSE curves for each of the six ANFIS' (1-6) after trial and error.



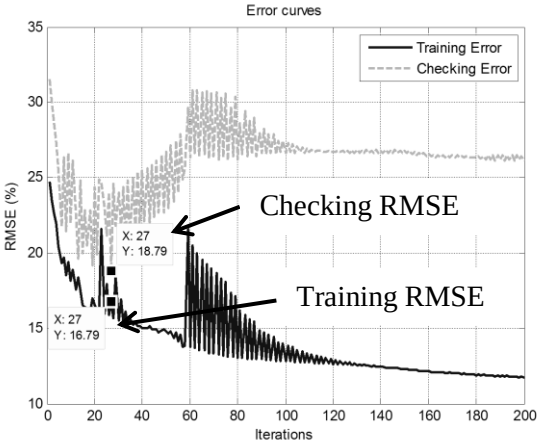
(1)



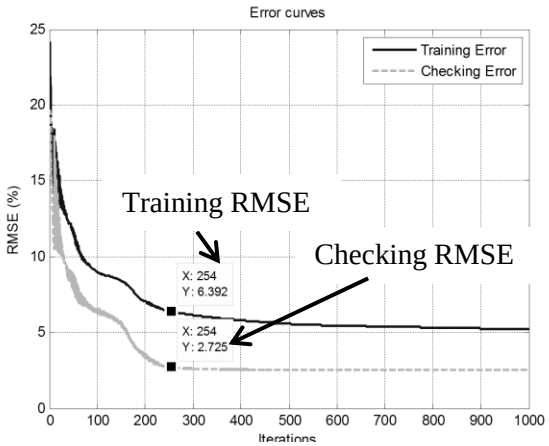
(2)



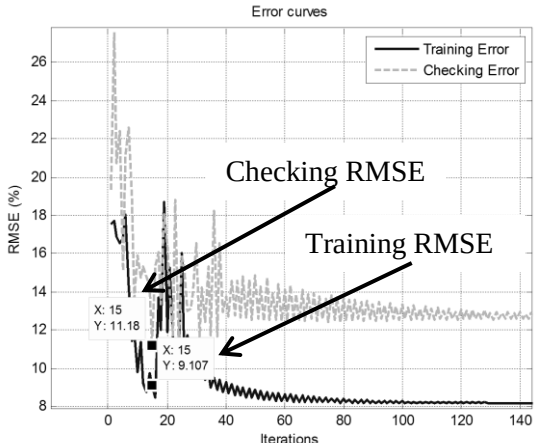
(3)



(4)



(5)



(6)

Figure 36: Trial 1 RMSE training and checking curves for the six ANFIS' (1-6) under three membership functions for each input

From trial and error in generating the six ANFIS', it can be observed that ANFIS 1 and 5 represents a convergence when the training RMSE curve flattens out and when the checking RMSE continues to decrease. The rest of the ANFIS' (2, 3, 4, and 6) represent convergence when the checking RMSE reaches a minimum before overfitting. Oscillations can be observed in ANFIS 3, 4, and 6. An attempt to decrease the initial step size for these ANFIS' was made to remove the oscillations. Table 6 shows the training and checking results for the six ANFIS'.

Table 6: Trial 1 training and checking results for the six ANFIS'

ANFIS	r_a	Initial step size	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	0.45	0.001	77	1.82	5.47
2	0.45	0.001	383	15.27	25.20
3	0.45	0.01	13	6.45	9.23
4	0.45	0.02	27	16.79	18.79
5	0.60	0.01	254	2.73	6.39
6	0.40	0.13	15	9.11	11.18

r_a -- User specified radius or range of influence for subtractive clustering to generate three membership functions for each input

Table 7 shows the classification results for the six ANFIS'. Classifications of the true positive specific heartbeats are denoted by TP. These heartbeats represent the normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. Classification of the true negative heartbeats without the specific heartbeat is denoted as TN. These beats are represented by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23.

Table 7: Trial 1 classification results for the six ANFIS'

ANFIS	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	175	100	100	100
2	32	158	90.48	65.31	98.14
3	34	175	99.52	100	99.43
4	34	175	99.52	100	99.43
5	34	175	99.52	100	99.43
6	34	175	99.52	100	99.43

TP – True positive specific heartbeats being classified. These are the six heartbeat types corresponding to the six ANFIS': normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN—True negative heartbeats without the specific heartbeat classified. These are represented through the six ANFIS' by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23.

Sample calculations from the second ANFIS for classification results in Table 7 using equations (68-70) where N here represents the total number of heartbeats tested:

$$Accuracy\ (\%) = \frac{TP+TN}{N}\ x\,100\% = \frac{32+158}{210}\ x\,100\% = 90.48\% \tag{71}$$

$$Sensitivity\ (\%) = \frac{TP}{TP+FN}\ x\,100\% = \frac{TP}{TP+(175-TN)}\ x\,100\% = \frac{32}{32+(175-158)}\ x\,100\% = 65.31\% \tag{72}$$

$$Specificity\ (\%) = \frac{TN}{TN+FP}\ x\,100\% = \frac{TN}{TN+(35-TP)}\ x\,100\% = \frac{158}{158+(35-32)}\ x\,100\% = 98.14\% \tag{73}$$

The results in Table 7 shows the ANFIS classifier is effective at detecting specific heartbeats. The second ANFIS was a poor classification between PVC heartbeats and all heartbeat types without the PVC heartbeats. This can be observed from the 90.48% classification and the sensitivity (measure of actual PVC heartbeats) of 65.31%. An analysis was done to observe why this was the case. The PVC heartbeats resemble that of the LBBB heartbeats for this trial. Table 8 shows how the input features of the PVC and LBBB heartbeats are similar in value. This can be shown in inputs 1, 3, and 5.

Table 8: Analysis of a PVC and LBBB heartbeat that were classified incorrectly

Input	1	2	3	4	5	6	7
PVC	0.20	0.32	1.42	0.053	1.04	0.69	0.73
LBBB	0.21	0.20	1.45	0.017	1.06	0.60	0.63

Two more trials were performed on random heartbeat records as discussed in the beginning of this section. Figures 37 and 38 shows the trials 2 and 3 training and checking RMSE curves for each of the six ANFIS'. Tables 9 and 11 shows the trials 2 and 3 training and checking results for the six ANFIS'. Tables 10 and 12 shows the trials 2 and 3 classification results for the six ANFIS'.

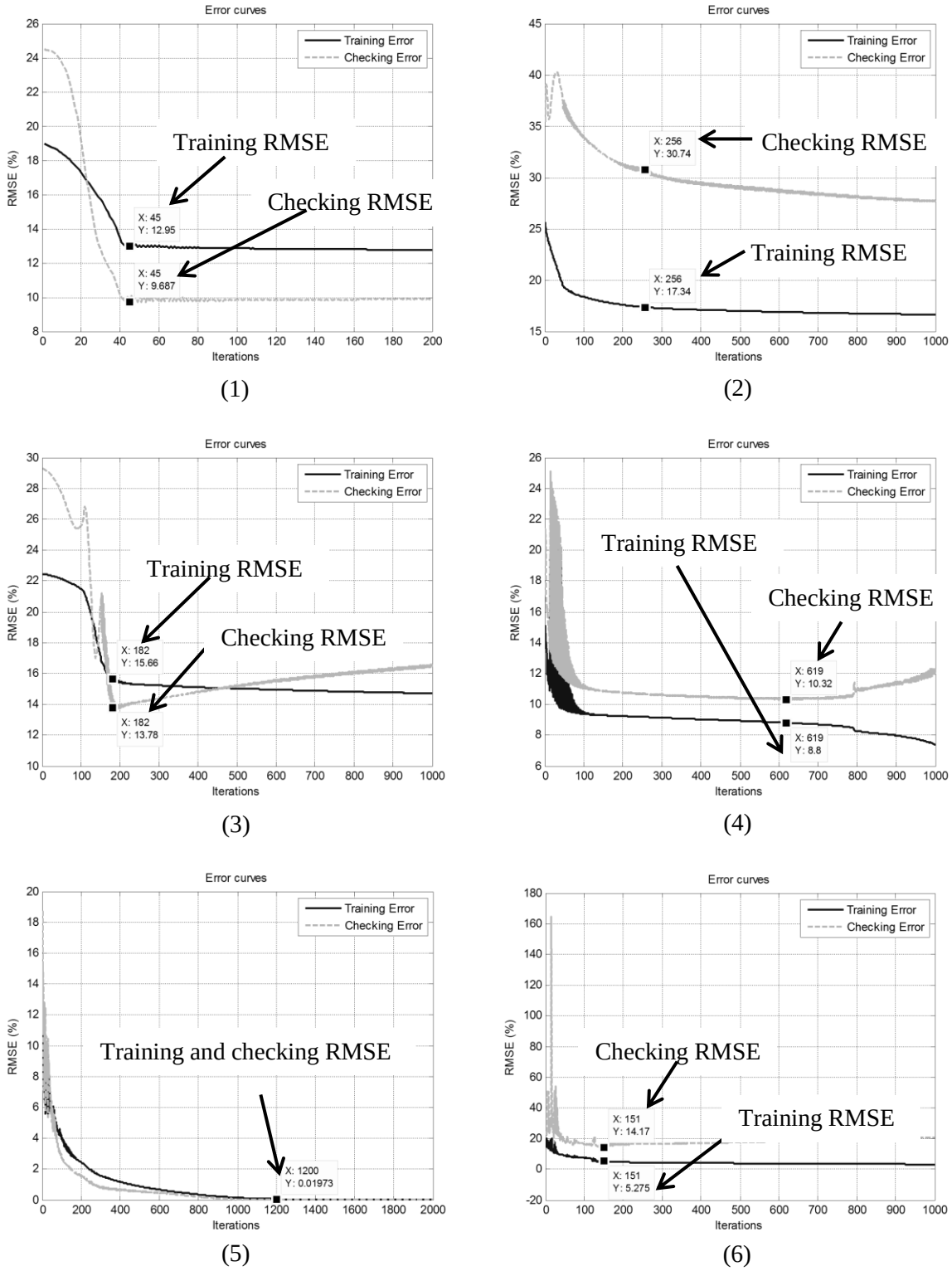


Figure 37: Trial 2 RMSE training and checking curves for the six ANFIS' (1-6) under three membership functions for each input

Table 9: Trial 2 training and checking results for the six ANFIS'

ANFIS	r_a	Initial step size	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	0.80	0.001	45	12.95	9.69
2	0.80	0.001	256	17.34	30.74
3	0.80	0.0001	182	15.66	13.78
4	0.90	0.03	619	8.80	10.32
5	0.90	0.05	1200	0.020	0.020
6	0.80	0.07	151	5.28	14.17

r_a -- User specified radius or range of influence for subtractive clustering to generate three membership functions for each input

Table 10: Trial 2 classification results for the six ANFIS'

ANFIS	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	173	99.05	94.59	100
2	25	169	92.38	80.65	94.41
3	31	175	98.10	100	97.77
4	32	173	97.62	94.12	98.30
5	35	175	100	100	100
6	34	174	99.05	97.14	99.43

TP – True positive specific heartbeats being classified. These are the six heartbeat types corresponding to the six ANFIS': normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats.

TN—True negative heartbeats without the specific heartbeat classified. These are represented through the six ANFIS' by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23.

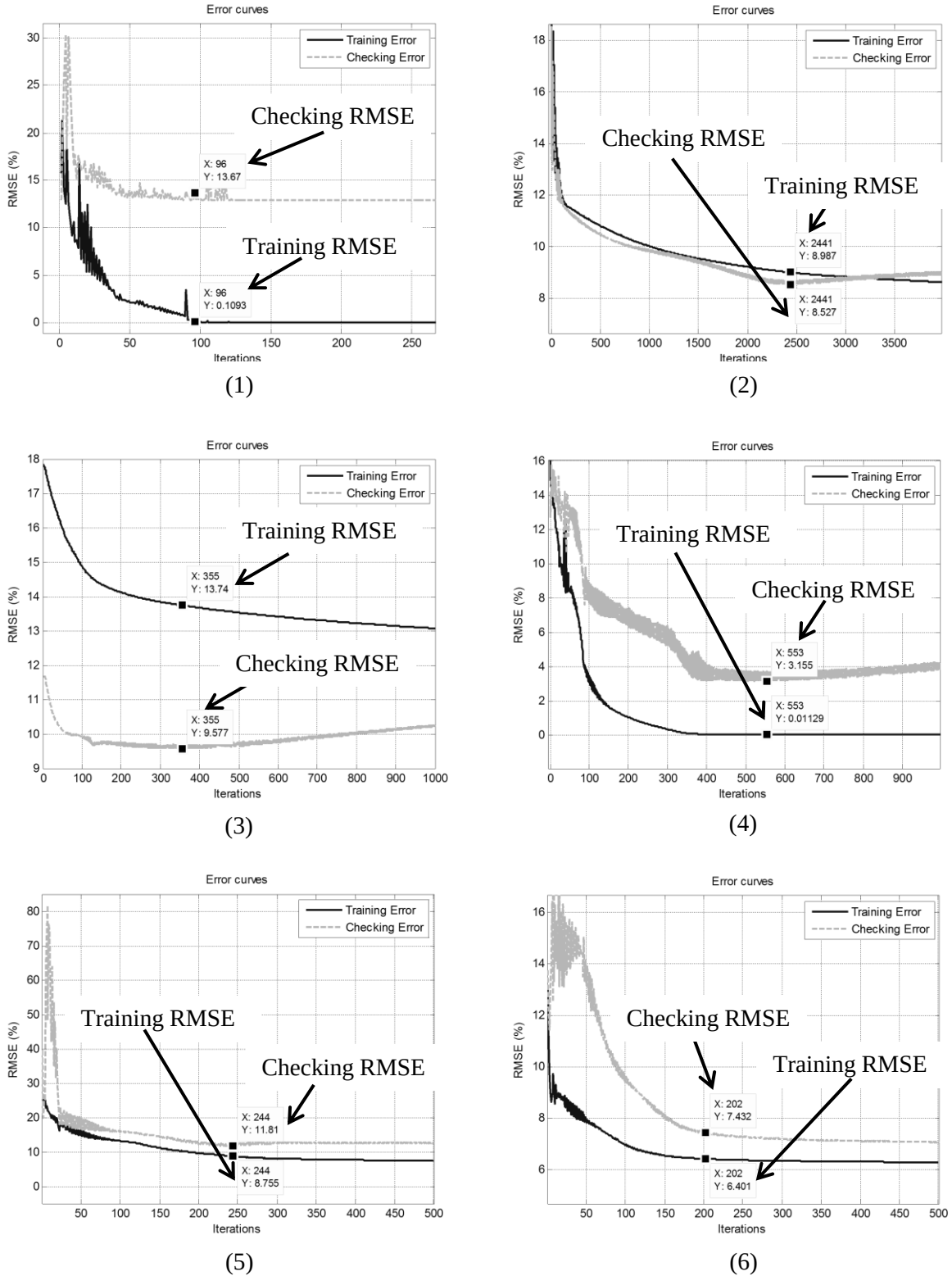


Figure 38: Trial 3 RMSE training and checking curves for the six ANFIS' (1-6) under three membership functions for each input

Table 11: Trial 3 training and checking results for the six ANFIS'

ANFIS	r_a	Initial step size	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	0.73	0.1	96	0.11	13.67
2	0.70	0.07	2441	8.99	8.53
3	0.80	0.001	355	13.74	9.58
4	0.80	0.1	553	0.011	3.16
5	0.90	0.1	244	8.76	11.81
6	0.70	0.03	202	6.40	7.43

r_a -- User specified radius or range of influence for subtractive clustering to generate three membership functions for each input

Table 12: Trial 3 classification results for the six ANFIS'

ANFIS	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	33	174	98.57	97.06	98.86
2	28	175	96.67	100	96.15
3	33	174	98.57	97.06	98.86
4	35	172	98.57	92.11	100
5	35	174	99.52	97.22	100
6	35	173	99.05	94.59	100

TP – True positive specific heartbeats being classified. These are the six heartbeat types corresponding to the six ANFIS': normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN—True negative heartbeats without the specific heartbeat classified. These are represented through the six ANFIS' by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23.

The average training RMSE for all three trials was calculated as 8.35%. The average training iterations calculated was about 405 iterations. The average run time to train each ANFIS coded in MATLAB for all trials was 2.83 seconds on a Windows 7 Intel Core i5 CPU running at 2.30 GHz.

Table 13 shows the average accuracies, sensitivies, and specificites for trials 1, 2, and 3. The average accuracy from the three trials was 98.10%. The average sensitivity from the three trials was 94.99%. The average specificity from the three trials was 98.87%.

Table 13: Average accuracy, sensitivity, and specificity results for 3 trials for subtractive clustering preprocessed ANFIS

Trial	Average Accuracy (%)	Average Sensitivity (%)	Average Specificity (%)
1	98.10	94.22	99.31
2	97.70	94.42	98.32
3	98.49	96.34	98.98

5.2: Justification of Chosen Input Records

There are several justifications for the choice of ECG records for the ANFIS classifier. First justification is the different heart rates from patient to patient. This makes it especially difficult to differentiate heartbeats of the same type from patient to patient. Different orientations of probe placement relative to the heart in measuring the ECG signals are another factor in the choice of input records [20]. Performing a 12-lead ECG analysis from patient to patient is bound to expose a difference in heartbeats of the same type. Movement noise is another factor for the choice of input records. It was discussed in [20] as well as observed that the ECG records exhibited movement noise over the course of thirty minutes. This would cause a difference in heartbeats of the same type.

In observing these factors, this thesis emphasizes the comparison between the proposed ANFIS classifier under subtractive clustering, two ANN algorithms, and ANFIS under grid partitioning. And for this reason, the classifiers simulated are patient specific. Patient to patient tests would have to be retrained to make sure the classifier is optimized for a specific patient. This thesis does not address the immediate problem of classifying heartbeats from patient to patient. As of now the doctor would have to aid in the training in order to allow for an effective patient to patient classification.

However, having classified six records of six different heartbeats proves the effectiveness of the ANFIS classifier. The proposed method discussed in Section 4.1 and the results discussed in Section 5.1 show the ANFIS classifier can be comparable to an ANN and the ANFIS under grid partitioning. For future improvements in addition to being able to classify patient to patient effectively, see Chapter 6.

5.3: Comparison to ANN

In comparing to the ANN, the same three trials as in Section 5.1 were performed. This means the same heartbeats were used for training, validating, and testing the classifiers. The ANN algorithms used for classification were the gradient descent backpropagation and Levenberg-Marquardt.

The ANN classifiers were tested through trial and error. One hidden layer was used, as it is was enough to perform classification. Hyperbolic tangent activation functions were chosen for the hidden neurons and one ouput neuron, as this type of output activation function has been proven effective for pattern recognition [26]. A logistic sigmoid has been considered to be more biologically realistic due to its range from '0' to '1', however, it runs into theoretical and experimental difficulties with certain types of computational problems. One of the difficulties is getting "stuck" during training. This is due to the fact that if a strongly-negative input is provided to the logistic-sigmoid, then its output values are near zero. A hyperbolic tangent function has strong negative inputs that will map to negative outputs. Additionally, only zero-valued inputs are mapped to near-zero outputs. These properties make the network less likely to get "stuck" during training [15]. The number of hidden neurons needed to be at least greater than seven (the number of inputs) for an effective classification. Both layers (hidden and output) have biases with corresponding weights that are updated. Figure 39 shows the neural network structure for both the gradient descent and Levenberg-Marquardt algorithms. From this figure, it can be shown that there are 56 weights and 8 biases that need to be updated. Initially the weights and biases are randomly set between -1 and 1.

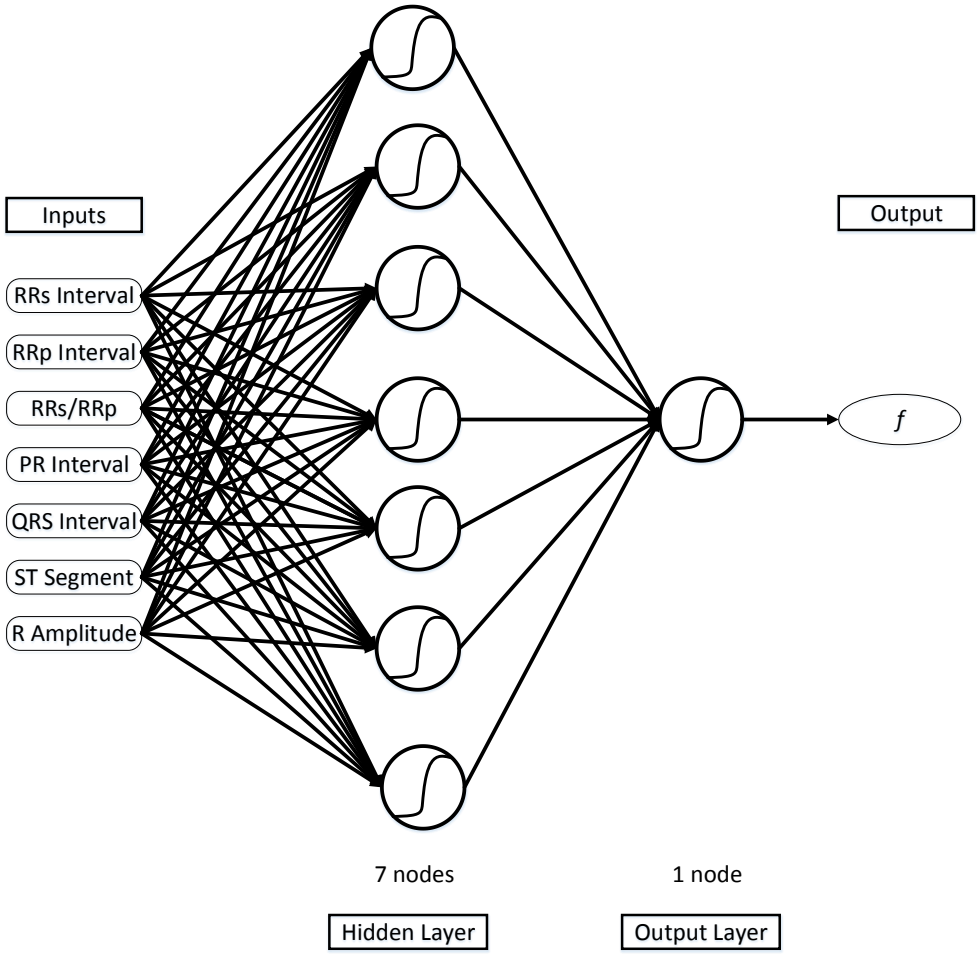


Figure 39: Neural network structure for gradient descent and Levenberg-Marquardt algorithms

The gradient descent network was trained, validated, and tested for all three trials. Trial and error in an attempt to classify the six heartbeats was performed. It was found that a learning rate of one for all six ANFIS' produced a reasonable training and validating result. An attempt to increase the performance was made by increasing the number of hidden neurons. The error curve reached the same convergence as when the hidden neuron count was seven. For this reason, the number of hidden neurons was kept at seven. The error curve of Figure 40 shows the performance result of the gradient descent algorithm for the training data of normal heartbeats N and all the heartbeats without the specific normal heartbeats $\overline{N}$. All heartbeats were correctly classified when the same testing data from Section 5.1 was presented to the trained gradient descent network.

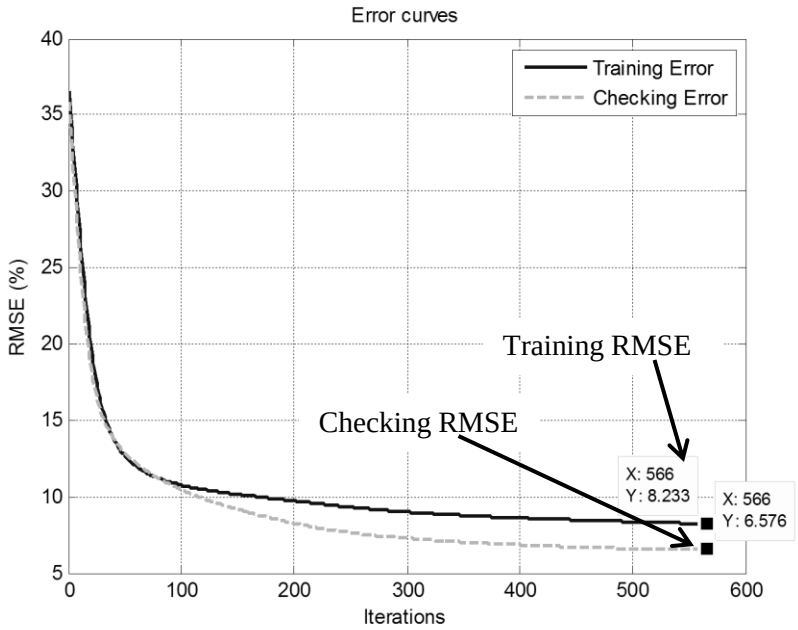


Figure 40: Trial 1 RMSE training and checking curves for the first ANFIS training and checking data for gradient descent. Convergence was reached at 566 iterations from overfitting (minimum checking RMSE reached).

Tables 14 and 15 shows the trial 1 training, checking, and classification results using the same trial 1 data used to train the six ANFIS' in Section 5.1. Tables16 and 17 shows trial 2 training, checking, and classification results. Tables 18 and 19 shows trial 3 training, checking, and classification results.

Table 14: Trial 1 training and checking results for the six ANFIS training and checking data for gradient descent

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	566	8.24	6.58
2	114	22.54	31.74
3	3584	7.95	5.67
4	177	30.14	30.16
5	188	25.18	24.67
6	707	10.16	6.25

Table 15: Trial 1 classification results for the six gradient descent neural networks (NN)

NN	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	175	100	100	100
2	30	146	83.81	50.85	96.69
3	31	175	98.10	100	97.77
4	35	171	98.10	89.74	100
5	32	171	96.67	88.89	98.28
6	34	175	99.52	100.00	99.43

TP -- Specific heartbeats being classified. These are the six heartbeat types corresponding to the six neural networks: normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN -- Heartbeats without the specific heartbeat classified. These are represented through the six neural networks by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23 (similar to the ANFIS').

Table 16: Trial 2 training and checking results for the six ANFIS training and checking data for gradient descent

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	314	25.34	28.26
2	430	25.86	24.40
3	659	15.90	13.76
4	136	13.00	17.77
5	338	12.19	20.80
6	65	13.61	18.86

Table 17: Trial 2 classification results for the six gradient descent neural networks (NN)

NN	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	32	172	97.14	91.43	98.29
2	16	169	88.10	72.73	89.89
3	32	170	96.19	86.49	98.27
4	33	167	95.24	80.49	98.82
5	35	170	97.62	87.50	100
6	32	173	97.62	94.12	98.30

TP -- Specific heartbeats being classified. These are the six heartbeat types corresponding to the six neural networks: normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN -- Heartbeats without the specific heartbeat classified. These are represented through the six neural networks by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23 (similar to the ANFIS').

Table 18: Trial 3 training and checking results for the six ANFIS training and checking data for gradient descent

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	82	17.42	14.31
2	239	23.71	22.43
3	1358	12.23	2.49
4	1551	6.90	8.57
5	263	27.82	29.03
6	1038	5.93	6.10

Table 19: Trial 3 classification results for the six gradient descent neural networks (NN)

NN	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	34	167	95.71	80.95	99.40
2	13	175	89.52	100	88.83
3	33	175	99.05	100	98.87
4	35	172	98.57	92.11	100.00
5	31	159	90.48	65.96	97.55
6	35	174	99.52	97.22	100.00

TP -- Specific heartbeats being classified. These are the six heartbeat types corresponding to the six neural networks: normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN -- Heartbeats without the specific heartbeat classified. These are represented through the six neural networks by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23 (similar to the ANFIS').

The averge training RMSE was calculated from all three trials as 16.90%. The average training iterations calculated was about 656 iterations. The average run time to train each gradient descent ANN coded in MATLAB for all trials was 1.44 seconds on a Windows 7 Intel Core i5 CPU running at 2.30 GHz.

Table 20 shows the average classification results for the three trials under the gradient descent. The average accuracy from the three trials was 95.61%. The average sensitivity from the three trials was 87.69%. The average specificity from the three trials was 97.80%.

Table 20: Average accuracy, sensitivity, and specificity results for 3 trials for gradient descent

Trial	Average Accuracy (%)	Average Sensitivity (%)	Average Specificity (%)
1	96.03	88.25	98.69
2	95.32	85.46	97.26
3	95.48	89.37	97.44

A Levenberg-Marquardt neural network, as discussed in Section 3.5, network was then trained, validated, and tested. Trial and error in an attempt to classify the six heartbeats was performed. The error curve of Figure 41 shows the performance result of the Levenberg-Marquardt algorithm for the training data of normal heartbeats N and all the heartbeats without the specific normal heartbeats $\overline{N}$. All heartbeats were correctly classified when the same testing data from Section 5.1 was presented to the trained Levenberg-Marquardt network. Note several of the trained neural networks needed to be trained at a lower learning rate than a learning rate of one. However, most trained neural networks were trained at a learning rate of one.

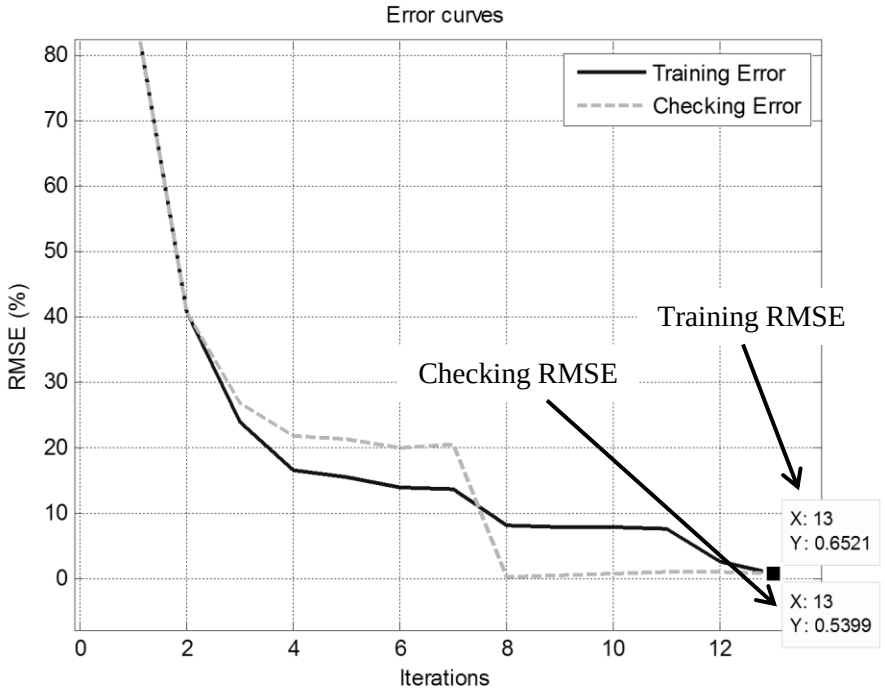


Figure 41: Trial 1 RMSE training and checking curves for the first ANFIS training and checking data for Levenberg-Marquardt. Convergence was reached at 13 iterations from overfitting (minimum checking RMSE reached).

Tables 21 and 22 shows the trial 1 training, checking, and classification results using the same trial 1 data used to train the six ANFIS' in Section 5.1. Tables 23 and 24 shows trial 2 training, checking, and classification results. Tables 25 and 26 shows trial 3 training, checking, and classification results.

Table 21: Trial 1 training and checking results for the six ANFIS training and checking data for Levenberg-Marquardt

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	13	0.65	0.54
2	10	12.20	33.02
3	13	5.83	10.53
4	13	13.11	33.53
5	21	1.62	4.50
6	13	0.12	0.12

Table 22: Trial 1 classification results for the six Levenberg-Marquardt neural networks (NN)

NN	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	175	100	100	100
2	30	171	95.71	88.24	97.16
3	31	175	98.10	100	97.77
4	30	171	95.71	88.24	97.16
5	35	173	99.05	94.59	100
6	35	173	99.05	94.59	100

TP -- Specific heartbeats being classified. These are the six heartbeat types corresponding to the six neural networks: normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats.

TN -- Heartbeats without the specific heartbeat classified. These are represented through the six neural networks by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23 (similar to the ANFIS').

Table 23: Trial 2 training and checking results for the six ANFIS training and checking data for Levenberg-Marquardt

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	14	7.89	18.78
2	19	0.040	18.29
3	22	10.27	14.20
4	18	0.039	12.94
5	17	0.078	10.10
6	15	0.066	15.11

Table 24: Trial 2 classification results for the six Levenberg-Marquardt neural networks (NN)

NN	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	33	175	99.05	100	98.87
2	30	167	93.81	78.95	97.09
3	31	171	96.19	88.57	97.71
4	32	173	97.62	94.12	98.30
5	35	174	99.52	97.22	100
6	35	175	100.00	100.00	100

TP -- Specific heartbeats being classified. These are the six heartbeat types corresponding to the six neural networks: normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats.

TN -- Heartbeats without the specific heartbeat classified. These are represented through the six neural networks by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23 (similar to the ANFIS').

Table 25: Trial 3 training and checking results for the six ANFIS training and checking data for Levenberg-Marquardt

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	15	0.060	12.91
2	21	0.065	0.43
3	21	5.51	8.07
4	16	0.074	0.074
5	20	0.021	0.021
6	12	0.16	0.16

Table 26: Trial 3 classification results for the six Levenberg-Marquardt neural networks (NN)

NN	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	174	99.52	97.22	100
2	28	175	96.67	100	96.15
3	34	173	98.57	94.44	99.43
4	35	173	99.05	94.59	100
5	35	172	98.57	92.11	100
6	35	175	100	100	100

TP -- Specific heartbeats being classified. These are the six heartbeat types corresponding to the six neural networks: normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN -- Heartbeats without the specific heartbeat classified. These are represented through the six neural networks by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23 (similar to the ANFIS').

The averge training RMSE was calculated from all three trials as 3.21%. The average training iterations calculated was about 16 iterations. The average run time to train each Levenberg-Marquardt ANN coded in MATLAB for all trials was 702 milliseconds on a Windows 7 Intel Core i5 CPU running at 2.30 GHz.

Table 27 shows the average classification results for the three trials under the Levenberg-Marquardt algorithm. The average accuracy from the three trials was 97.98%. The average sensitivity from the three trials was 94.09%. The average specificity from the three trials was 98.81%.

Table 27: Average accuracy, sensitivity, and specificity results for 3 trials for Levenberg-Marquardt

Trial	Average Accuracy (%)	Average Sensitivity (%)	Average Specificity (%)
1	97.94	94.28	98.68
2	97.43	91.77	98.62
3	98.57	96.23	99.12

5.4: Comparison to ANFIS under Grid Partitioning

The six ANFIS' from Section 5.1 was then trained, validated, and tested under grid partitioning. As discussed in Section 4.3, two triangle membership functions were chosen for each of the six ANFIS'. The triangle membership function was described in Section 2.5 under equation (8). The same three trials of data used in Section 5.1 are used in this section. Figure 42 shows the training and checking RMSE curves of the first ANFIS for the first trial. Figure 43 shows the initial FIS generated from grid partitioning. The range of each input is determined by minimum and maximum input feature in their corresponding input feature range. The figure shows first (solid function) and second membership functions (dashed function) linguistically as 'low' and 'high' partitions respectively for each input feature. Figure 44 shows the FIS after training the first ANFIS.

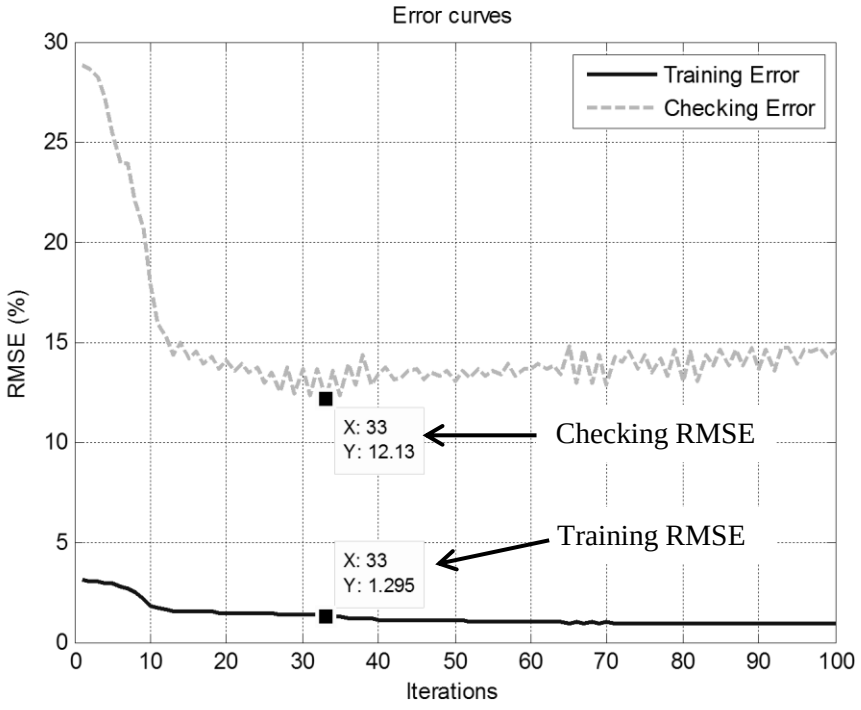


Figure 42: Trial 1 RMSE training and checking curves for the first ANFIS under two membership functions grid partitioned for each input for 100 iterations. Initial step size is 0.01. Convergence was reached at 33 iterations (minimum checking error before overfitting).

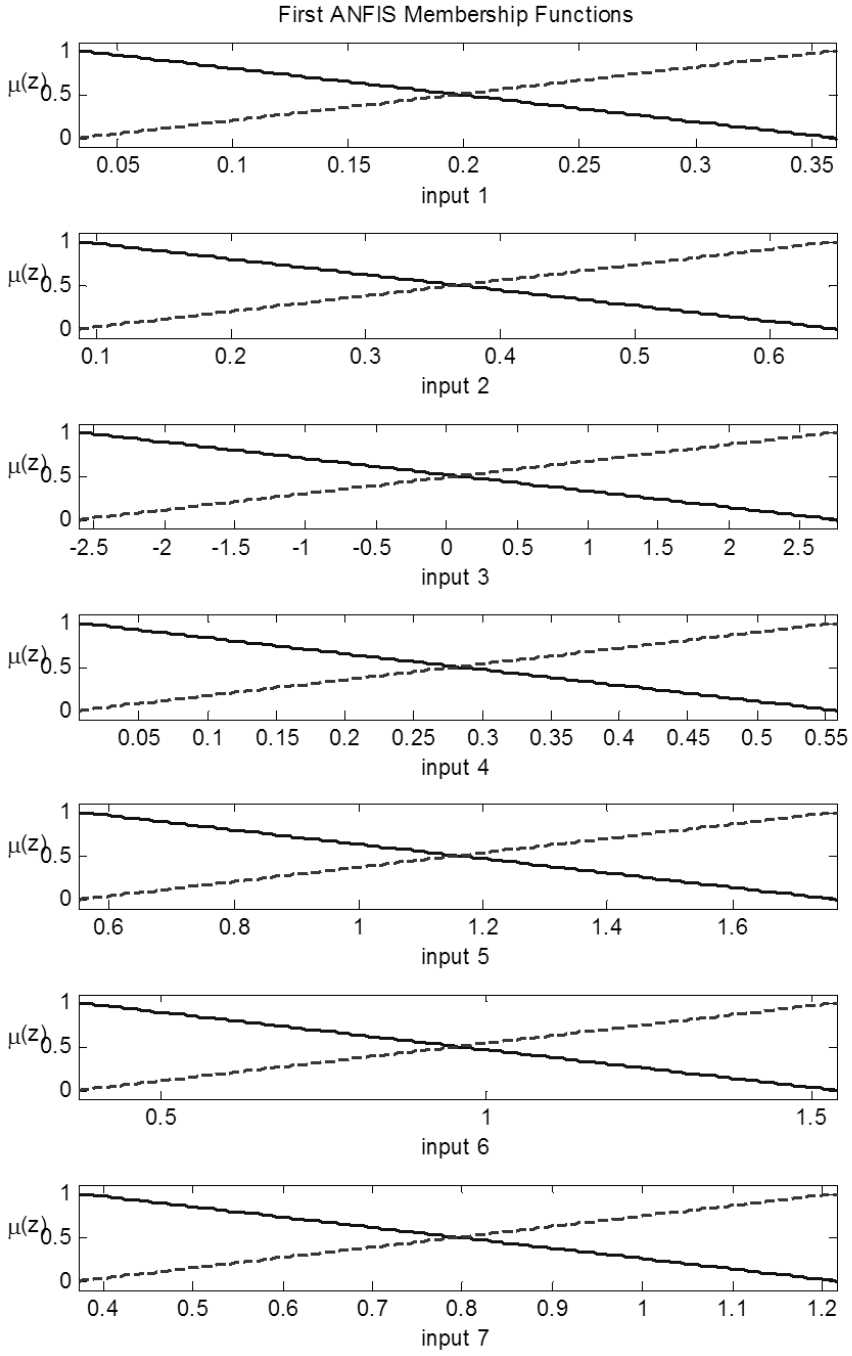


Figure 43: Trial 1 initial grid partition FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function.

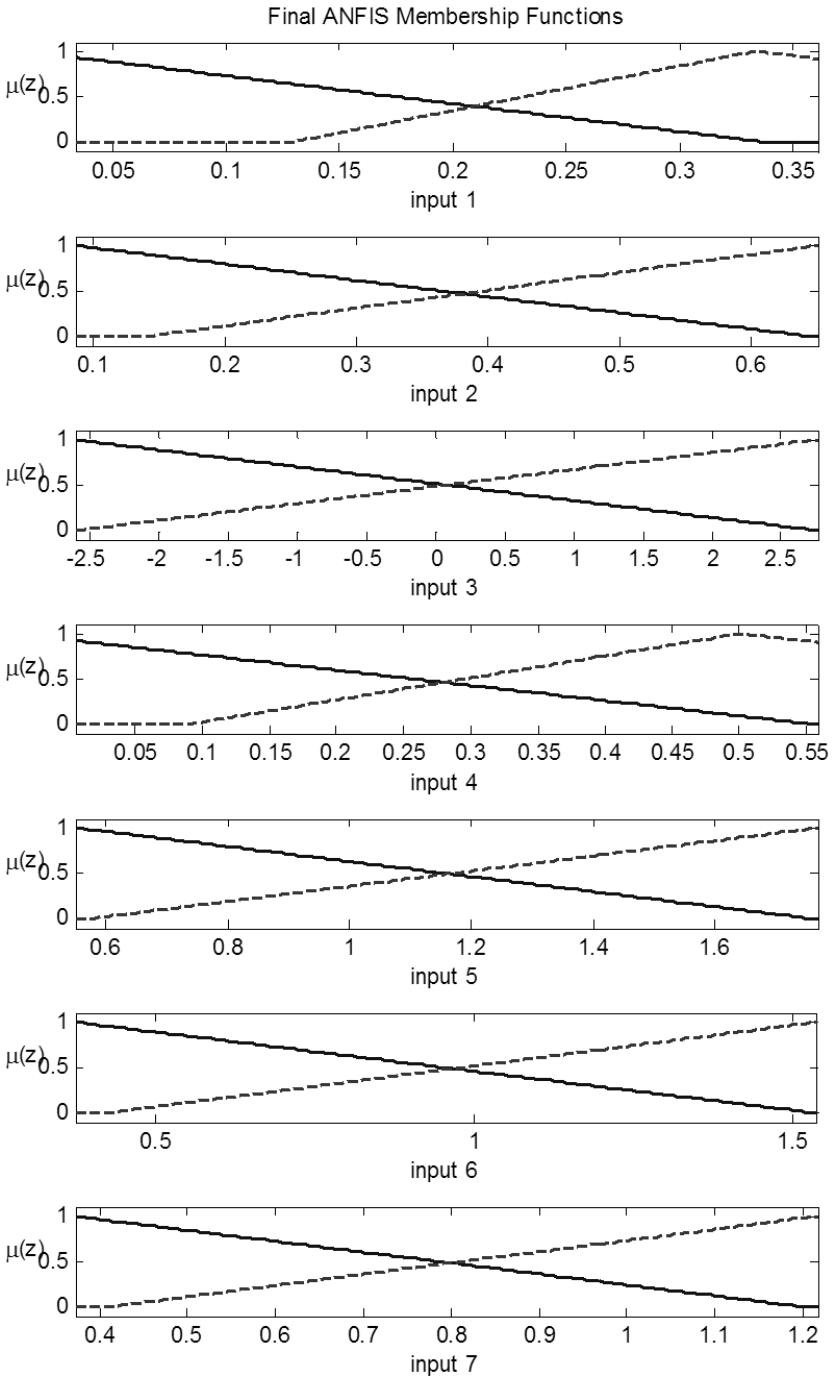


Figure 44: Trial 1 adapted FIS for the first ANFIS. The solid dark function represents the first membership function. The dashed function represents the second membership function.

An example of the first input 1 (QRS Interval) membership function parameters and a set of consequent parameters before and after training are shown in Figure 45.

Antecedent Parameters:

$$\{a, b, c\} = \{-0.294, 0.0333, 0.361\}$$

$$\downarrow$$

$$\{a, b, c\} = \{-0.294, 0.0117, 0.339\}$$

Consequent Parameters:

$$\{q_{11}, q_{21}, q_{31}, q_{41}, q_{51}, q_{61}, q_{71}, r_1\} = \{0, 0, 0, 0, 0, 0, 0, 0\}$$

$$\downarrow$$

$$\{q_{11}, q_{21}, q_{31}, q_{41}, q_{51}, q_{61}, q_{71}, r_1\} = \{5.96, -9.92, -1.11, -9.22, 0.04, 3.36, -1.50, -0.51\}$$

Figure 45: Antecedent and consequent parameters before and after training for grid partion FIS

Due to both the exponential complexity of grid partitioning for seven inputs and the run time to train each ANFIS, the initial step size remained at 0.01. Tables 28 and 29 shows the first trial training, checking, and classification results for the six ANFIS' under grid partitioning two triangle membership functions.

Table 28: Trial 1 training and checking results for the six grid partitioned ANFIS'

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	33	1.30	12.13
2	18	8.05	39.46
3	15	3.51	18.46
4	16	10.05	82.80
5	42	7.55	93.79
6	39	1.33	39.68

Table 29: Trial 1 classification results for the six grid partitioned ANFIS'

ANFIS	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	173	99.05	94.59	100
2	32	140	81.90	47.76	97.90
3	31	172	96.67	91.18	97.73
4	22	164	88.57	66.67	92.66
5	35	169	97.14	85.37	100
6	30	175	97.62	100	97

TP – True positive specific heartbeats being classified. These are the six heartbeat types corresponding to the six ANFIS': normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN—True negative heartbeats without the specific heartbeat classified. These are represented through the six ANFIS' by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23.

From Tables 28 it can be shown there is a high checking RMSE at convergence compared to the training and checking results in previous algorithms. Table 29 shows two classification results that are noteworthy for discussion. Specifically, the second and fourth ANFIS resulted in a poor classification. The second ANFIS classified between PVCs and the other five heartbeats without the PVC heartbeats. The fourth ANFIS classified between LBBBs and the other five heartbeats without the LBBB heartbeats. As discussed in Section 5.1 under the same data used to train, validate, and test the six ANFIS' that the PVCs are being misclassified as LBBBs as well as the LBBBs being misclassified as PVCs. The second and third trials show similar results. Tables 30 and 31 shows the second trial training, checking, and classification results. Tables 32 and 33 shows the third trial training, checking, and classification results.

Table 30: Trial 2 training and checking results for the six grid partitioned ANFIS'

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	17	4.65	52.86
2	100	2.56	58.84
3	87	3.71	31.19
4	12	1.89	42.48
5	43	1.41	33.95
6	39	1.17	27.97

Table 31: Trial 2 classification results for the six grid partitioned ANFIS'

ANFIS	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	162	93.81	72.92	100
2	27	165	91.43	72.97	95.38
3	28	168	93.33	80.00	96.00
4	32	169	95.71	84.21	98.26
5	35	170	97.62	87.50	100
6	34	172	98.10	91.89	99.42

TP – True positive specific heartbeats being classified. These are the six heartbeat types corresponding to the six ANFIS': normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN—True negative heartbeats without the specific heartbeat classified. These are represented through the six ANFIS' by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23.

Table 32: Trial 3 training and checking results for the six grid partitioned ANFIS'

ANFIS	Training iterations	Training RMSE (%)	Checking RMSE (%)
1	1	3.80	77.11
2	1	5.10	47.79
3	27	3.44	33.78
4	1	5.54	51.31
5	99	4.10	67.92
6	78	0.28	9.63

Table 33: Trial 3 classification results for the six grid partitioned ANFIS'

ANFIS	TP	TN	Accuracy (%)	Sensitivity (%)	Specificity (%)
1	35	170	97.62	87.50	100
2	27	173	95.24	93.10	95.58
3	31	165	93.33	75.61	97.63
4	32	172	97.14	91.43	98.29
5	35	155	90.48	63.64	100
6	35	173	99.05	94.59	100

TP – True positive specific heartbeats being classified. These are the six heartbeat types corresponding to the six ANFIS': normal (N), PVC (V), APC (A), LBBB (L), RBBB (R), and paced (P) heartbeats. TN—True negative heartbeats without the specific heartbeat classified. These are represented through the six ANFIS' by $\overline{N}$, $\overline{V}$, $\overline{A}$, $\overline{L}$, $\overline{R}$, and $\overline{P}$ in Figure 23.

The average training RMSE for all three trials was calculated as 3.86%. The average training iterations calculated was about 37 iterations. The average run time to train each ANFIS coded in MATLAB for all trials was 15.27 minutes on a Windows 7 Intel Core i5 CPU running at 2.30 GHz.

Table 34 shows the average accuracies, sensitivies, and specificites for trials 1, 2, and 3. The average accuracy from the three trials was 94.59%. The average sensitivity from the three trials was 82.64%. The average specificity from the three trials was 97.90%.

Table 34: Average accuracy, sensitivity, and specificity results for 3 trials for grid partitioned ANFIS

Trial	Average Accuracy (%)	Average Sensitivity (%)	Average Specificity (%)
1	93.49	80.93	97.58
2	95.24	83.32	97.81
3	95.05	83.67	98.30

Three other types of membership functions were then performed for grid partitioning the seven inputs. These functions were gaussian, gaussian bell, and trapezoid. These functions were described in Section 2.5 under equations (6, 7, and 9). Again two membership functions were performed for each input in training, validating, and testing the six ANFIS'. The average training RMSE, average training iterations, and average run time was similar to the grid partitioned ANFIS' under the triangle membership functions. Table 35 shows the average accuracies, sensitivies, and specificites for three trials for each of type of membership function. Due to similar average accuracies, sensitivities, and specificies for the different membership function types, it can be concluded that there is no particular effective membership function type. However, clearly the trapezoid membership function type performed at a slightly higher average accuracy due to having the highest number of antecedent parameters (four parameters).

Table 35: Different membership function type average accuracy, sensitivity, and specificity results for 3 trials for grid partitioned ANFIS

Membership Function	Average Accuracy (%)	Average Sensitivity (%)	Average Specificity (%)
Triangle	94.59	82.64	97.90
Gaussian	94.60	85.79	96.99
Gaussian Bell	94.44	83.58	97.29
Trapezoid	94.84	88.14	97.23

CHAPTER 6: CONCLUSIONS AND FUTURE WORKS

The ANFIS had the advantage of integrating the best features of fuzzy systems and neural networks in ECG classificiation. The fuzzy system was able to represent prior knowledge into a set of constraints to reduce the optimization search space. Consequently, fuzzy systems provided smoothness from the interpolation among the rules. The neural network was able to adapt through backpropagation to automate the fuzzy parametric tuning. Table 36 shows the classification results of the four algorithms performed for three trials under the same data used to train, validate, and test for randomized ECG record selection and randomized heartbeats. ANFIS under subtractive clustering has a higher performance in terms of average accuracy, average sensitivity, and average specificity.

Table 36: Average accuracy, sensitivity, and specificity results for 3 trials for several algorithms

Algorithm	Average Accuracy (%)	Average Sensitivity (%)	Average Specificity (%)
ANFIS under subtractive clustering	98.10	94.99	98.87
Gradient Descent ANN	95.61	87.69	97.80
Levenberg-Marquardt ANN	97.98	94.09	98.81
ANFIS under grid partitioning (Triangle)	94.59	82.64	97.90

The ANFIS under subtractive clustering converged faster than the gradient descent ANN. However, the Levenberg-Marquardt ANN convergence was much faster even though the computational requirements are much higher per iteration. The higher convergence speed was because the algorithm is comprised of both the Guass-Newton method and gradient descent. As discussed in Section 3.5, Gauss-Newton method converges rapidly near a local or global minimum, but may also diverge. The gradient descent assures convergence through proper selection of the step size parameter but convergence is slow [22].

The average run time of the ANFIS under subtractive clustering was reasonable compared to the the two ANN algorithms even though it was performing two algorithms: subtractive clustering and

ANFIS. Both made use of LSE while the ANFIS performed an additional algorithm being the gradient descent. It is clear that the ANFIS under grid partititoning underperforms as far as convergence speed.

ANFIS has strong computational complexity restrictions. One way to reduce the complexity is integrating “don’t care” values in rules. This means an elimination of a connection is done between the fuzzification layer and the rule layer. It was also observed that because of large number of rules in grid partitioning, there was an increase in oscillations particularly for the checking error.

In comparing the performance of classifiers described in the literature review of Chapter 3, the ANFIS was shown to be an effective classifier. In fact, the method used to create a multiclass SVM in [10] is related to the proposed method of Figure 23 and Section 4.1. This is because both the ANFIS and SVM are binary classifiers.

In terms of updating the antecedent parameters, further speedup learning is possible using variants of the gradient method or other optimization techniques. Several include the conjugate gradient descent, second-order backpropagation, quick propagation, etc. In terms of updating the consequent parameters, further speedup learning is possible by implementing the Widrow-Hoff Least-Mean-Square algorithm. The algorithm is computationally efficient by parallel hardware implementation. However, it has been proven to converge slower than the LSE [15].

Future work on improving the ANFIS classifier could be done by analyzing the rules. Having an advantage of tuning rules over black box systems such as ANNs, the user could measure the degree of fulfillment for several rules. This was shown in Figure 1 and done in [7]. This could bring about a more effective FIS for evaluating test data.

Another possible future work is expanding the range of input features. Temporal and amplitude features were enough to bring about reasonable results. However, the analysis of classification could be expanded by introducing frequency domain features. In particular, it has been shown that discrete wavelet transform (DWT) or Lyapunov exponents could extract coefficients that effectively discriminate between different heartbeats [10], [38].

REFERENCES

[1] Abbas K., A., & Bassam, R. (2009). *Phonocardiography Signal Processing.* Morgan & Claypool.

[2] al., F. e. (n.d.). A Patient-Adaptive Profiling Scheme for ECG Beat Classification. IEEE Transactions of Information Technology in Biomedicine, VOL. 14, No. 5, September 2010.

[3] Edla, S., Kovvali, N., & Papandreou-Suppappola, A. (2014). *Electrocardiogram Signal Modeling With Adaptive Parameter Estimation Using Sequential Bayesian Methods.* New York: IEEE Transactions on Signal Processing.

[4] Amari, S. &. (1999). *Improving support vector machine classifiers by modifying kernel functions.*

[5] Cadogan, M. (2014). *ECG Basics*. Retrieved December 8, 2014, from Life in the Fastlane: http://lifeinthefastlane.com/ecg-library/basics/q-wave/

[6] Chandramoulseeswarean, S., Ahmed, H. M., & Samsuri, F. (2012). *Wavelet Diagnosis of ECG Signals with Kaiser Based Noise Diminution.* Kuantan, Malaysia: Journal of Biomedical Science an Engineering.

[7] Chikh A., M., Ammar, M., Marouf, & Radja, M. (2010, April 5). A Neuro-Fuzzy Identification of ECG Beats. Tlemcen, Algeria.

[8] Editors: Witold Pedrycz, A. (2012). *ECG Signal Processing, Classification and Interpretation: A Comprehensive Framework of Computational Intelligence.* New York City: Springer-Verlag London Limited 2012.

[9] FavoritePlus. (2013, May 22). *Different Types of ECG Machines.* Retrieved December 2, 2014, from FavoritePlus: Innovative Healthcare Products: http://www.favoriteplus.com/blog/types-ecg-machines/

[10] Fazihudeen, S., & P.V, S. (2013). *ECG Beat Classification Using Wavelets and SVM.* Department of Electronics and Communication Engineering, KMCT College of Engineering, Calicuit, India.

[11] GB, M., RG, M., Database, T. i.-B., & 11446209), 4.-5. (.-J. (n.d.). Retrieved July 2014, from http://www.physionet.org/physiobank/database/mitdb/

[12] Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing Third Edition.* Upper Saddle River, New Jersey: Pearson Education, Inc.

[13] Gregg, E. R., & Zhou Huai, S. (2013). *Patent No. 643,114.* United States of America.

[14] Inan Guler, E. D. (2005, April 15). Adaptive neuro-fuzzy inference system for classification of EEG signals using wavelet coefficients, Journal of Neuroscience Methods. Ankara, Turkey.

[15] Jang, J.-S. R. (1993). *ANFIS: Adaptive-Network-Based Fuzzy Inference System.* IEEE Transactions on Systems, Man, and Cybernetics, VOL. 23, NO. 3.

[16] Jang, J.-S. R. (1997). *Neuro-Fuzzy and Soft Computing.* Upper Sadle River, NJ: Prentice-Hall, Inc.

[17] Khazaee, A. (2013). *Automated Cardiac Beat Classification Using RBF Neural Networks.* Islamic Azad University, Department of Electrical Engineering, Bojnourd Branch, Bojnourd, Iran.

[18] Manab Kumar Das, S. A. (2014). ECG Signal Classification Technique using S-transform, BFO and LMS. Roukela, Department of Electronics and Communication Engineering National Instute of Technology, India.

[19] MathWorks. (2014). *Matlab Documentation Center.* Retrieved September 20, 2014, from MathWorks: http://www.mathworks.com/help/

[20] MIT Beth Israel Deaconess Medical Center (BIH). (2005). MIT-BIH Arrhythmia Database. Boston, Massachusetts, United States of America.

[21] Pan, J., & Tompkins J, W. (1985). *A Real-Time QRS Detection Algorithm.* IEEE Transactions of Biomedical Engineering, VOL. BME-32, NO. 3.

[22] Gothwal, H., Kedawat, S., & Kumar, R. (2011). *Cardiac Arrythmias Detection in an ECG Beat Signal Using Fast Fourier Transform and Artificial Neural Network.* Jaipur, India: Journal of Biomedical Science and Engineering.

[23] Ryan, S. S. (2014, October). *Understanding the Electrocadriogram (EKG or ECG) Signal.* Retrieved December 2, 2014, from Atrial Fibrillation Resources for Patients: http://a-fib.com/treatments-for-atrial-fibrillation/diagnostic-tests/the-ekg-signal/

[24] S, C., J, S., N, S., B, H., & E., L. (1990, March). *R- and S-wave amplitude changes with acute anterior transmural myocardial ischemia. Correlations with left ventricular filling pressures.* Retrieved December 7, 2014, from US National Library of Medicine National Institutes of Health: http://www.ncbi.nlm.nih.gov/pubmed/2306959

[25] Shorten, G., & Burke, M. (2011). *A Time Domain Based Classifier for ECG Pattern Recognition.* Boston, Massachusetts: 33rd Anual International Conference of the IEEE EMBS.

[26] The Math Works, Inc. (2012). *Fuzzy Logic Tool Box: Design and simulate fuzzy logic systems.* Retrieved October 18, 2013, from MathWorks: http://www.mathworks.com/products/datasheets/pdf/fuzzy-logic-toolbox.pdf

[27] The Seven Countries Study. (2014). *The Seven Countries Study*. Retrieved December 4, 2014, from ECG Findings and Corornary Heart Disease: http://sevencountriesstudy.com/ecg-predictors-and-coronary-heart-disease

[28] Y. Hu, S. P. (1997, September). "A patient-adaptable ECG beat classifier using a mixture of experts approach," IEEE Trans. Biomed. Eng., vol. 44, no. 9, pp. 891-900.

[29] Yu, D. X.-H. (2015). EE 509 Lecture 1: Wk_1_T_Intro_Natural_Computation. California Polytechnic State University, San Luis Obispo, California, United States of America.

[30] Chiu, S. L. (1994). *Fuzzy Model Identification Based on Cluster Estimation.* Thousand Oaks, California: Rockwell Science Center.

[31] Ranade, A. (1990). *A simpler analysis of the Karp-Zhang parallel branch-and-bound method.* Berkely, California: Computer Science Division, University of California, Berkely.

[32] Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms.* New York: Plenum Press.

[33] Yager, R. R., & Filev, D. P. (1994). *Essentials of fuzzy modeling and control* . John Wiley & Sons, Inc.

[34] Bataineh, K. M., Naji, M., & Saqer, M. (2011). *A Comparison Study between Various Fuzzy Clustering Algoirthms.* Department of Mechanical Engineering. Irbid, Jordan: Jordan University of Science and Technology.

[35] Romera, M. M., Vazquez, M. A., & Garcia, J. C. (2007). *Comparing Improved.* University of ALcala, Electronics. Berlin: Springer-Verlag Heidelberg.

[36] Eftekhari, M., & Katebi, S. (2008). *Extracting compact fuzzy rules for nonlinear system modeling using subtractive clustering, GA and unscented filter.* School of Engineering, Computer Science and Engineering Department. Shiraz, Iran: Elsevier.

[37] Sumathi, S., Beaulah, H. L., & Vanithamani, R. (2014). *A Wavelet Transform Based Feature Extraction and Classifiation of Cardiac Disorder.* New York: Springer Science+Business Media.

[38] Ubeyli, D. E. (2008). *Adaptive neuro-fuzzy inference system for classification of ECG signals using Lyapunov exponents.* Ankara, Turkey: Elsevier BV.

APPENDICES

APPENDIX A – List of Acronyms

ANN – Artificial Neural Network
ECG or EKG – Electrocardiograph(y)
MIT-BIH – Massachusetts Institute of Technology-Beth Israel Hospital
WFDB – Waveform Database Toolbox
FIS – Fuzzy Inference System
ANFIS – Adaptive Neuro-Fuzzy Inference System
SVM – Support Vector Machine
LMS – Least Mean Square(s)
LSE – Least Squares Estimator
RBF – Radial Basis Function
MLP – Multi-layer Perceptron
SWT – Stationary Wavelet Transform
DWT – Discrete Wavelet Transform
FCM – Fuzzy C-means Algorithm
VCG – Vectorcardiography
EMG – Electromyography
EEG – Electroencephalography
CI – Computational Intelligence
PVC – Premature Ventricular Contraction
APC – Atrial Premature Ventricular Contraction
LBBB – Left Bundle Branch Block
RBBB – Right Bundle Branch Block

APPENDIX B – MATLAB Code

Plot ECG Signal from MIT-BIH Arrhythmia Database with Annotations:

```
function [yt, awaves, a_type,num] = WFDB_QRS(filename, num);
% This script uses the WFDB (Waveform Database) Toolbox from Physionet

% Input paramters (arguments) are:
%   filename from MIT-Arrhythmia database

% Output values returned are:
%   Annotations for input layer of classifier

% Revised: 10/18/14 - by Brad Funsten
%          11/10/14 - Switched input type from annotation file to ecgpuwave
%          type file for not only normal beats, but abnormal as well.
%          1/6/15 - Integrated script as a function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% clear all;
% close all;
% clc;
num = 1;
fs = 360; % sample rate for MIT-BIH Arrhytmia Database
samples = 646400; % samples for 30 minutes

%filename = 'mitdb/207';
[tm, signal]=rdsamp(filename, 1, samples); % first signal--double precision
                                           % (64 bits)
%[tm, signal]=rdsamp(filename,[2], samples); % second signal--
                                           %double precision (64 bits)
%[tm, signal]=rdsamp(filename,[],[],[],2); % single precision (32 bits)

signal = signal(:,1);

% Time axis
horizontal_axis = 0:1:size(signal,2) - 1;
time = horizontal_axis .* 1 / fs;

% LPF of signal
[zt, num] = l_or_hpf(signal, 30, 3, fs, time, 'low', num);

% HPF of signal to detrend (remove baseline shift)
[yt, num] = l_or_hpf(zt, 1, 3, fs, time, 'high', num);

plot(yt);
num = num + 1;
grid on;
xlabel('Samples');
ylabel('Amplitude (mV)');
title(filename);

% r peak
[ann,type,~,~]=rdann(filename,'atr');
```

```
for k = 1 : size(ann,1)
  if ann(k) <= samples
      stop = k;
  end
end

ann = ann(1 : stop);
type = type(1 : stop);
hold on;
plot(ann, yt(ann),'rv','MarkerFaceColor','y');

% Type of beat
text(ann, yt(ann) + 0.2, type);

% Extract PQRST information:
% Create qrs file through sqrs function
sqrs(filename, 'qrs');
% Create test file through ecgpuwave function
ecgpuwave(filename,'test',[],[],'qrs');

% p wave
pwaves=rdann(filename,'test',[],[],[],'p');

for k = 1 : size(pwaves,1)
  if pwaves(k) <= samples
      stop = k;
  end
end

pwaves = pwaves(1 : stop);
hold on;
plot(pwaves, yt(pwaves), 'ro', 'MarkerFaceColor', 'b');

% t wave
[twaves,t_type,t_subtype,t_chan,t_num]=rdann(filename,'test',[],[],[],'t');

for k = 1 : size(twaves,1)
  if twaves(k) <= samples
      stop = k;
  end
end

twaves = twaves(1 : stop);
hold on;
plot(twaves, yt(twaves),'bv', 'MarkerFaceColor', 'c');

% onset
[onset,o_type,o_subtype,o_chan,o_num]=rdann(filename,'test',[],[],[],'(');

for k = 1 : size(onset,1)
  if onset(k) <= samples
      stop = k;
  end
```

```
end

onset = onset(1 : stop);
hold on;
plot(onset, yt(onset),'rs','MarkerFaceColor','k');

% offset
[offset,f_type,f_subtype,f_chan,f_num]=rdann(filename,'test',[],[],[],')');

for k = 1 : size(offset,1)
  if offset(k) <= samples
      stop = k;
  end
end

offset = offset(1 : stop);

hold on;
plot(offset, yt(offset),'yo', 'MarkerFaceColor', 'm');


% beat type
[beat,b_type,b_subtype,b_chan,b_num]=rdann(filename,'test',[],[],[],'N');

for k = 1 : size(beat,1)
  if beat(k) <= samples
      stop = k;
  end
end

beat = beat(1 : stop);

hold on;
plot(beat, yt(beat),'y^', 'MarkerFaceColor', 'b');

% Overall legend
legend('ECG Signal', 'R','P','T', 'Onset', 'Offset','beat');

% Input layer
[awaves,a_type,a_subtype,a_chan,a_num]=rdann(filename,'test',[],[],[]);

% Replace annotation type characters with ecgpuwave type (a_type)
% characters
ann(1) = [];
type(1) = [];

j = 1;
for i = 1 : length(a_type)

    if j == length(ann)
      break;
    end

    if (a_type(i) == 'N')
```

```
      while(ann(j) < (awaves(i) - 100))
          j = j + 1;
          if j == length(ann)
              break;
          end
      end

      a_type(i) = type(j);
    end
end
awaves = awaves(awaves <= 646400);
a_type = a_type(1:length(awaves));
```

ECG Input Layer:

```
function [inputN, inputV, inputA, inputL, inputR, inputP, num] =
input_layer_NVALRP(yt, awaves, a_type, num);
% This script makes use of WFDB_QRS script and determines the input layer
% to the Artificial Neuro-Fuzzy Inference

% Input paramters (arguments) are:
%   MIT-BIH ECG signal with annotation information

% Output values returned are:
%   Input layer for ANFIS

% Revised: 11/9/14 - by Brad Funsten
%           3/27/15 - Added all six heartbeats to extract input features.
%           There are 15 input features extracted.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num = 2;
snum = 6;
%close all;

% Delete the first heart beat
lengthSig = length(a_type);
for i = 1 : lengthSig
    if a_type(i) == 'N' || a_type(i) == 'V' || a_type(i) == 'A' ||...
            a_type(i) == 'L' || a_type(i) == 'R' || a_type(i) == '/'
        a_type(i) = [];
        awaves(i) = [];
        break;
    end
end

% Delete the last heart beat
lengthSig = length(a_type);
for i = lengthSig:-1:1
    if a_type(i) == 'N' || a_type(i) == 'V' || a_type(i) == 'A' ||...
            a_type(i) == 'L' || a_type(i) == 'R' || a_type(i) == '/'
        a_type(i) = [];
        awaves(i) = [];
        break;
    end
end

%% QRS interval
figure(num); num = num + 1;

% QRS interval for normal beats
if find(a_type == 'N')
N = find(a_type == 'N');
QN = N - 1;
SN = N + 1;
QN_sample = awaves(QN);
SN_sample = awaves(SN);
QRS_N = (SN_sample - QN_sample) ./ 360;
subplot(snum,1,1);
```

```
% QRSi=.04:.008:.12;
% bar(QRSi,histc(QRS_N,QRSi),'hist');
hist(QRS_N);
grid on;
title('Histogram of Normal QRS intervals');
ylabel('# of bins');
xlabel('seconds');
end

% QRS interval for PVC beats
if find(a_type == 'V')
V = find(a_type == 'V');
QV = V - 1;
SV = V + 1;
QV_sample = awaves(QV);
SV_sample = awaves(SV);
QRS_V = (SV_sample - QV_sample) ./ 360;
subplot(snum,1,2);
%QRSi=.04:.008:.12;
%bar(QRSi,histc(QRS_R,QRSi),'hist');
hist(QRS_V);
grid on;
title('Histogram of PVC QRS intervals');
ylabel('# of bins');
xlabel('seconds');
end

% QRS interval for APC beats
if find(a_type == 'A')
A = find(a_type == 'A');
QA = A - 1;
SA = A + 1;
QA_sample = awaves(QA);
SA_sample = awaves(SA);
QRS_A = (SA_sample - QA_sample) ./ 360;
subplot(snum,1,3);
% bar(QRSi,histc(QRS_A,QRSi),'hist');
hist(QRS_A);
grid on;
title('Histogram of APC QRS intervals');
ylabel('# of bins');
xlabel('seconds');
end

% QRS interval for LBBB beats
if find(a_type == 'L')
L = find(a_type == 'L');
QL = L - 1;
SL = L + 1;
QL_sample = awaves(QL);
SL_sample = awaves(SL);
QRS_L = (SL_sample - QL_sample) ./ 360;
subplot(snum,1,4);
%bar(QRSi,histc(QRS_L,QRSi),'hist');
hist(QRS_L);
grid on;
```

```
title('Histogram of LBBB QRS intervals');
ylabel('# of bins');
xlabel('seconds');
end

% QRS interval for RBBB beats
if find(a_type == 'R')
R = find(a_type == 'R');
QR = R - 1;
SR = R + 1;
QR_sample = awaves(QR);
SR_sample = awaves(SR);
QRS_R = (SR_sample - QR_sample) ./ 360;
subplot(snum,1,5);
%bar(QRSi,histc(QRS_R,QRSi),'hist');
hist(QRS_R);
grid on;
title('Histogram of RBBB QRS intervals');
ylabel('# of bins');
xlabel('seconds');
end

% QRS interval for Paced beats
if find(a_type == '/')
P = find(a_type == '/');
QP = P - 1;
SP = P + 1;
QP_sample = awaves(QP);
SP_sample = awaves(SP);
QRS_R = (SP_sample - QP_sample) ./ 360;
subplot(snum,1,6);
%bar(QRSi,histc(QRS_R,QRSi),'hist');
hist(QRS_R);
grid on;
title('Histogram of Paced QRS intervals');
ylabel('# of bins');
xlabel('seconds');
end

%% PR interval (P wave onset to R wave onset (Q onset))
figure(num); num = num + 1;

% Normal beat PR interval
if find(a_type == 'N')
PNO = N - 4;
PRN = (awaves(QN) - awaves(PNO)) ./ 360;
subplot(snum,1,1);
% PRi=0:.05:.5;
% bar(PRi,histc(PRN,PRi),'hist');
hist(PRN);
grid on;
title('Histogram of Normal PR intervals');
ylabel('# of bins');
xlabel('seconds');
end
```

```
% PVC beat PR interval
if find(a_type == 'V')
PVO = V - 4;
PRV = (awaves(QV) - awaves(PVO)) ./ 360;
subplot(snum,1,2);
% bar(PRi,histc(PRV,PRi),'hist');
hist(PRV);
grid on;
title('Histogram of PVC PR intervals');
ylabel('# of bins');
xlabel('seconds');
end

% APC beat PR interval
if find(a_type == 'A')
PAO = A - 4;
PRA = (awaves(QA) - awaves(PAO)) ./ 360;
subplot(snum,1,3);
%bar(PRi,histc(PRA,PRi),'hist');
hist(PRA);
grid on;
title('Histogram of APC PR intervals');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB beat PR interval
if find(a_type == 'L')
PLO = L - 4;
PRL = (awaves(QL) - awaves(PLO)) ./ 360;
subplot(snum,1,4);
% PRi=0:.05:.5;
% bar(PRi,histc(PRL,PRi),'hist');
hist(PRL);
grid on;
title('Histogram of LBBB PR intervals');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB beat PR interval
if find(a_type == 'R')
PRO = R - 4;
PRR = (awaves(QR) - awaves(PRO)) ./ 360;
subplot(snum,1,5);
% PRi=0:.05:.5;
% bar(PRi,histc(PRR,PRi),'hist');
hist(PRR);
grid on;
title('Histogram of RBBB PR intervals');
ylabel('# of bins');
xlabel('seconds');
end

% Paced beat PR interval
if find(a_type == '/')
```

```
PPO = P - 4;
PRP = (awaves(QP) - awaves(PPO)) ./ 360;
subplot(snum,1,6);
% PRi=0:.05:.5;
% bar(PRi,histc(PRP,PRi),'hist');
hist(PRP);
grid on;
title('Histogram of Paced PR intervals');
ylabel('# of bins');
xlabel('seconds');
end

%% PR segment (P wave offset to R wave onset (Q onset))
figure(num); num = num + 1;

% Normal beat PR interval
if find(a_type == 'N')
PNF = N - 2;
PRN_seg = (awaves(QN) - awaves(PNF)) ./ 360;
subplot(snum,1,1);
% PR_seg_i=0:.02:.2;
% bar(PR_seg_i,histc(PRN_seg,PR_seg_i),'hist');
hist(PRN_seg);
grid on;
title('Histogram of Normal PR segments');
ylabel('# of bins');
xlabel('seconds');
end

% PVC beat PR interval
if find(a_type == 'V')
PVF = V - 2;
PRV_seg = (awaves(QV) - awaves(PVF)) ./ 360;
subplot(snum,1,2);
% bar(PR_seg_i,histc(PRV_seg,PR_seg_i),'hist');
hist(PRV_seg);
grid on;
title('Histogram of PVC PP segments');
ylabel('# of bins');
xlabel('seconds');
end

% APC beat PR interval
if find(a_type == 'A')
PAF = A - 2;
PRA_seg = (awaves(QA) - awaves(PAF)) ./ 360;
subplot(snum,1,3);
%bar(PR_seg_i,histc(PRA_seg,PR_seg_i),'hist');
hist(PRA_seg);
grid on;
title('Histogram of APC PR segments');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB beat PR interval
```

```
if find(a_type == 'L')
PLF = L - 2;
PRL_seg = (awaves(QL) - awaves(PLF)) ./ 360;
subplot(snum,1,4);
% PR_seg_i=0:.02:.2;
% bar(PR_seg_i,histc(PRL_seg,PR_seg_i),'hist');
hist(PRL_seg);
grid on;
title('Histogram of LBBB PR segments');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB beat PR interval
if find(a_type == 'R')
PRF = R - 2;
PRR_seg = (awaves(QR) - awaves(PRF)) ./ 360;
subplot(snum,1,5);
% PR_seg_i=0:.02:.2;
% bar(PR_seg_i,histc(PRR_seg,PR_seg_i),'hist');
hist(PRR_seg);
grid on;
title('Histogram of RBBB PR segments');
ylabel('# of bins');
xlabel('seconds');
end

% Paced beat PR interval
if find(a_type == '/')
PPF = P - 2;
PRP_seg = (awaves(QP) - awaves(PPF)) ./ 360;
subplot(snum,1,6);
% PR_seg_i=0:.02:.2;
% bar(PR_seg_i,histc(PRP_seg,PR_seg_i),'hist');
hist(PRP_seg);
grid on;
title('Histogram of Paced PP segments');
ylabel('# of bins');
xlabel('seconds');
end


%% P wave

figure(num); num = num + 1;

% Normal beat P wave
if find(a_type == 'N')
PN_sample = (awaves(PNF) - awaves(PNO)) ./ 360;
subplot(snum,1,1);
% P_sample_i=0:.035:.35;
% bar(P_sample_i,histc(PN_sample,P_sample_i),'hist');
hist(PN_sample);
grid on;
title('Histogram of Normal P wave intervals');
ylabel('# of bins');
```

```
xlabel('seconds');
end

% PVC beat PP interval
if find(a_type == 'V')
PV_sample = (awaves(PVF) - awaves(PVO)) ./ 360;
subplot(snum,1,2);
% bar(P_sample_i,histc(PV_sample,P_sample_i),'hist');
hist(PV_sample);
grid on;
title('Histogram of PVC P wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% APC beat PR interval
if find(a_type == 'A')
PA_sample = (awaves(PAF) - awaves(PAO)) ./ 360;
subplot(snum,1,3);
% bar(P_sample_i,histc(PA_sample,P_sample_i),'hist');
hist(PA_sample);
grid on;
title('Histogram of APC P wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB beat P wave
if find(a_type == 'L')
PL_sample = (awaves(PLF) - awaves(PLO)) ./ 360;
subplot(snum,1,4);
% P_sample_i=0:.035:.35;
% bar(P_sample_i,histc(PL_sample,P_sample_i),'hist');
hist(PL_sample);
grid on;
title('Histogram of LBBB P wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB beat P wave
if find(a_type == 'R')
PR_sample = (awaves(PRF) - awaves(PRO)) ./ 360;
subplot(snum,1,5);
% P_sample_i=0:.035:.35;
% bar(P_sample_i,histc(PR_sample,P_sample_i),'hist');
hist(PR_sample);
grid on;
title('Histogram of RBBB P wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% Paced beat P wave
if find(a_type == '/')
PP_sample = (awaves(PPF) - awaves(PPO)) ./ 360;
```

```
subplot(snum,1,6);
% P_sample_i=0:.035:.35;
% bar(P_sample_i,histc(PP_sample,P_sample_i),'hist');
hist(PP_sample);
grid on;
title('Histogram of Paced P wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

%% P amplitude

figure(num); num = num + 1;

% Normal beat P amplitude
if find(a_type == 'N')
PN = awaves(N - 3);
PampN = yt(PN);
subplot(snum,1,1);
% Pamp_i=-.2:.04:.2;
% bar(Pamp_i,histc(PampN,Pamp_i),'hist');
hist(PampN);
grid on;
title('Histogram of Normal P amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% PVC beat P amplitude
if find(a_type == 'V')
PV = awaves(V - 3);
PampV = yt(PV);
subplot(snum,1,2);
% bar(Pamp_i,histc(PampV,Pamp_i),'hist');
hist(PampV);
grid on;
title('Histogram of PVC P amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% APC beat P amplitude
if find(a_type == 'A')
PA = awaves(A - 3);
PampA = yt(PA);
subplot(snum,1,3);
%bar(Pamp_i,histc(PampA,Pamp_i),'hist');
hist(PampA);
grid on;
title('Histogram of APC P amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% LBBB beat P amplitude
if find(a_type == 'L')
```

```
PL = awaves(L - 3);
PampL = yt(PL);
subplot(snum,1,4);
% Pamp_i=-.2:.04:.2;
% bar(Pamp_i,histc(PampL,Pamp_i),'hist');
hist(PampL);
grid on;
title('Histogram of LBBB P amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% RBBB beat P amplitude
if find(a_type == 'R')
PR = awaves(R - 3);
PampR = yt(PR);
subplot(snum,1,5);
% Pamp_i=-.2:.04:.2;
% bar(Pamp_i,histc(PampR,Pamp_i),'hist');
hist(PampR);
grid on;
title('Histogram of RBBB P amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% Paced beat P amplitude
if find(a_type == '/')
PP = awaves(P - 3);
PampP = yt(PP);
subplot(snum,1,6);
% Pamp_i=-.2:.04:.2;
% bar(Pamp_i,histc(PampP,Pamp_i),'hist');
hist(PampP);
grid on;
title('Histogram of Paced P amplitudes');
ylabel('# of bins');
xlabel('mV');
end

%% R amplitude

figure(num); num = num + 1;

% Normal beat R amplitude
if find(a_type == 'N')
RN = awaves(N);
RampN = yt(RN);
subplot(snum,1,1);
% Ramp_i=-2.5:.41:2.6;
% bar(Ramp_i,histc(RampN,Ramp_i),'hist');
hist(RampN);
grid on;
title('Histogram of Normal R amplitudes');
ylabel('# of bins');
xlabel('mV');
```

```
end

% PVC beat P amplitude
if find(a_type == 'V')
RV = awaves(V);
RampV = yt(RV);
subplot(snum,1,2);
% bar(Ramp_i,histc(RampV,Ramp_i),'hist');
hist(RampV);
grid on;
title('Histogram of PVC R amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% APC beat R amplitude
if find(a_type == 'A')
RA = awaves(A);
RampA = yt(RA);
subplot(snum,1,3);
% bar(Ramp_i,histc(RampA,Ramp_i),'hist');
hist(RampA);
grid on;
title('Histogram of APC R amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% LBBB beat R amplitude
if find(a_type == 'L')
RL = awaves(L);
RampL = yt(RL);
subplot(snum,1,4);
% Ramp_i=-2.5:.41:2.6;
% bar(Ramp_i,histc(RampL,Ramp_i),'hist');
hist(RampL);
grid on;
title('Histogram of LBBB R amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% RBBB beat R amplitude
if find(a_type == 'R')
RR = awaves(R);
RampR = yt(RR);
subplot(snum,1,5);
% Ramp_i=-2.5:.41:2.6;
% bar(Ramp_i,histc(RampR,Ramp_i),'hist');
hist(RampR);
grid on;
title('Histogram of RBBB R amplitudes');
ylabel('# of bins');
xlabel('mV');
end
```

```
% Paced beat R amplitude
if find(a_type == '/')
RP = awaves(P);
RampP = yt(RP);
subplot(snum,1,6);
% Ramp_i=-2.5:.41:2.6;
% bar(Ramp_i,histc(RampP,Ramp_i),'hist');
hist(RampP);
grid on;
title('Histogram of Paced R amplitudes');
ylabel('# of bins');
xlabel('mV');
end

%% Q onset amplitude

figure(num); num = num + 1;

% Normal beat Q onset amplitude
if find(a_type == 'N')
QampN = yt(QN_sample);
subplot(snum,1,1);
% Qamp_i=-.25:.03:.05;
% bar(Qamp_i,histc(QampN,Qamp_i),'hist');
hist(QampN);
grid on;
title('Histogram of Normal Q onset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% PVC beat Q onset amplitude
if find(a_type == 'V')
QampV = yt(QV_sample);
subplot(snum,1,2);
% bar(Qamp_i,histc(QampV,Qamp_i),'hist');
hist(QampV);
grid on;
title('Histogram of PVC Q onset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% APC beat Q onset amplitude
if find(a_type == 'A')
QampA = yt(QA_sample);
subplot(snum,1,3);
% bar(Qamp_i,histc(QampA,Qamp_i),'hist');
hist(QampA);
grid on;
title('Histogram of APC Q onset amplitudes');
ylabel('# of bins');
xlabel('mV');
end
```

```
% LBBB beat Q onset amplitude
if find(a_type == 'L')
QampL = yt(QL_sample);
subplot(snum,1,4);
% Qamp_i=-.25:.03:.05;
% bar(Qamp_i,histc(QampL,Qamp_i),'hist');
hist(QampL);
grid on;
title('Histogram of LBBB Q onset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% RBBB beat Q onset amplitude
if find(a_type == 'R')
QampR = yt(QR_sample);
subplot(snum,1,5);
% Qamp_i=-.25:.03:.05;
% bar(Qamp_i,histc(QampR,Qamp_i),'hist');
hist(QampR);
grid on;
title('Histogram of RBBB Q onset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% Paced beat Q onset amplitude
if find(a_type == '/')
QampP = yt(QP_sample);
subplot(snum,1,6);
% Qamp_i=-.25:.03:.05;
% bar(Qamp_i,histc(QampP,Qamp_i),'hist');
hist(QampP);
grid on;
title('Histogram of Paced Q onset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

%% S offset amplitude

figure(num); num = num + 1;

% Normal beat S offset amplitude
if find(a_type == 'N')
SampN = yt(SN_sample);
subplot(snum,1,1);
% Samp_i=-1.2:.04:0.4;
% bar(Samp_i,histc(SampN,Samp_i),'hist');
hist(SampN);
grid on;
title('Histogram of Normal S offset amplitudes');
ylabel('# of bins');
xlabel('mV');
end
```

```
% PVC beat S offset amplitude
if find(a_type == 'V')
SampV = yt(SV_sample);
subplot(snum,1,2);
% bar(Samp_i,histc(SampV,Samp_i),'hist');
hist(SampV);
grid on;
title('Histogram of PVC S offset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% APC beat S offset amplitude
if find(a_type == 'A')
SampA = yt(SA_sample);
subplot(snum,1,3);
%bar(Samp_i,histc(SampA,Samp_i),'hist');
hist(SampA);
grid on;
title('Histogram of APC S offset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% LBBB beat S offset amplitude
if find(a_type == 'L')
SampL = yt(SL_sample);
subplot(snum,1,4);
% Samp_i=-1.2:.04:0.4;
% bar(Samp_i,histc(SampL,Samp_i),'hist');
hist(SampL);
grid on;
title('Histogram of LBBB S offset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% RBBB beat S offset amplitude
if find(a_type == 'R')
SampR = yt(SR_sample);
subplot(snum,1,5);
% Samp_i=-1.2:.04:0.4;
% bar(Samp_i,histc(SampR,Samp_i),'hist');
hist(SampR);
grid on;
title('Histogram of RBBB S offset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

% Paced beat S offset amplitude
if find(a_type == '/')
SampP = yt(SP_sample);
subplot(snum,1,6);
% Samp_i=-1.2:.04:0.4;
% bar(Samp_i,histc(SampP,Samp_i),'hist');
```

```
hist(SampP);
grid on;
title('Histogram of Paced S offset amplitudes');
ylabel('# of bins');
xlabel('mV');
end

%% T wave

% remove t character from a_type
new_a_type = a_type;
for i = length(awaves):-1:1
    if a_type(i) == 't'
        new_a_type(i) = [];
    end
end

figure(num); num = num + 1;

% Normal beat T wave
if find(a_type == 'N')
TNO = N + 2;
TNF = N + 3;
TN_sample = (awaves(TNF) - awaves(TNO)) ./ 360;
subplot(snum,1,1);
% T_sample_i=0:.016:.16;
% bar(T_sample_i,histc(TN_sample,T_sample_i),'hist');
hist(TN_sample);
grid on;
title('Histogram of Normal T wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% PVC beat T wave
if find(a_type == 'V')
TVO = V + 2;
TVF = V + 3;
TV_sample = (awaves(TVF) - awaves(TVO)) ./ 360;
subplot(snum,1,2);
% bar(T_sample_i,histc(TV_sample,T_sample_i),'hist');
hist(TV_sample);
grid on;
title('Histogram of PVC T wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% APC beat T wave
if find(a_type == 'A')
TAO = A + 2;
TAF = A + 3;
TA_sample = (awaves(TAF) - awaves(TAO)) ./ 360;
subplot(snum,1,3);
%bar(T_sample_i,histc(TA_sample,T_sample_i),'hist');
hist(TA_sample);
```

```
grid on;
title('Histogram of APC T wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB beat T wave
if find(a_type == 'L')
TLO = L + 2;
TLF = L + 3;
TL_sample = (awaves(TLF) - awaves(TLO)) ./ 360;
subplot(snum,1,4);
% T_sample_i=0:.016:.16;
% bar(T_sample_i,histc(TL_sample,T_sample_i),'hist');
hist(TL_sample);
grid on;
title('Histogram of LBBB T wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB beat T wave
if find(a_type == 'R')
TRO = R + 2;
TRF = R + 3;
TR_sample = (awaves(TRF) - awaves(TRO)) ./ 360;
subplot(snum,1,5);
% T_sample_i=0:.016:.16;
% bar(T_sample_i,histc(TR_sample,T_sample_i),'hist');
hist(TR_sample);
grid on;
title('Histogram of RBBB T wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

% Paced beat T wave
if find(a_type == '/')
TPO = P + 2;
TPF = P + 3;
TP_sample = (awaves(TPF) - awaves(TPO)) ./ 360;
subplot(snum,1,6);
% T_sample_i=0:.016:.16;
% bar(T_sample_i,histc(TP_sample,T_sample_i),'hist');
hist(TP_sample);
grid on;
title('Histogram of Paced T wave intervals');
ylabel('# of bins');
xlabel('seconds');
end

%% ST interval (beginning of the S wave to the end of the T wave)

figure(num); num = num + 1;
```

```
% Normal beat ST interval
if find(a_type == 'N')
STN_sample = (awaves(TNF) - awaves(SN)) ./ 360;
subplot(snum,1,1);
% ST_sample_i=0:.02:1.4;
% bar(ST_sample_i,histc(STN_sample,ST_sample_i),'hist');
hist(STN_sample);
grid on;
title('Histogram of Normal ST intervals');
ylabel('# of bins');
xlabel('seconds');
end

% PVC beat ST interval
if find(a_type == 'V')
STV_sample = (awaves(TVF) - awaves(SV)) ./ 360;
subplot(snum,1,2);
%bar(ST_sample_i,histc(STV_sample,ST_sample_i),'hist');
hist(STV_sample);
grid on;
title('Histogram of PVC ST intervals');
ylabel('# of bins');
xlabel('seconds');
end

% APC beat ST interval
if find(a_type == 'A')
STA_sample = (awaves(TAF) - awaves(SA)) ./ 360;
subplot(snum,1,3);
% bar(ST_sample_i,histc(STA_sample,ST_sample_i),'hist');
hist(STA_sample);
grid on;
title('Histogram of APC ST intervals');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB beat ST interval
if find(a_type == 'L')
STL_sample = (awaves(TLF) - awaves(SL)) ./ 360;
subplot(snum,1,4);
% ST_sample_i=0:.02:1.4;
% bar(ST_sample_i,histc(STL_sample,ST_sample_i),'hist');
hist(STL_sample);
grid on;
title('Histogram of LBBB ST intervals');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB beat ST interval
if find(a_type == 'R')
STR_sample = (awaves(TRF) - awaves(SR)) ./ 360;
subplot(snum,1,5);
% ST_sample_i=0:.02:1.4;
% bar(ST_sample_i,histc(STR_sample,ST_sample_i),'hist');
```

```
hist(STR_sample);
grid on;
title('Histogram of RBBB ST intervals');
ylabel('# of bins');
xlabel('seconds');
end

% Paced beat ST interval
if find(a_type == '/')
STP_sample = (awaves(TPF) - awaves(SP)) ./ 360;
subplot(snum,1,6);
% ST_sample_i=0:.02:1.4;
% bar(ST_sample_i,histc(STP_sample,ST_sample_i),'hist');
hist(STP_sample);
grid on;
title('Histogram of Paced ST intervals');
ylabel('# of bins');
xlabel('seconds');
end

%% QT interval (beginning of the Q wave to the end of the T wave)

figure(num); num = num + 1;

% Normal beat QT interval
if find(a_type == 'N')
QTN_sample = (awaves(TNF) - awaves(QN)) ./ 360;
subplot(snum,1,1);
% QT_sample_i=.1:.025:1.6;
% bar(QT_sample_i,histc(QTN_sample,QT_sample_i),'hist');
hist(QTN_sample);
grid on;
title('Histogram of Normal QT intervals');
ylabel('# of bins');
xlabel('seconds');
end

% PVC beat QT interval
if find(a_type == 'V')
QTV_sample = (awaves(TVF) - awaves(QV)) ./ 360;
subplot(snum,1,2);
% bar(QT_sample_i,histc(QTV_sample,QT_sample_i),'hist');
hist(QTV_sample);
grid on;
title('Histogram of PVC QT intervals');
ylabel('# of bins');
xlabel('seconds');
end

% APC beat QT interval
if find(a_type == 'A')
QTA_sample = (awaves(TAF) - awaves(QA)) ./ 360;
subplot(snum,1,3);
% bar(QT_sample_i,histc(QTA_sample,QT_sample_i),'hist');
hist(QTA_sample);
grid on;
```

```
title('Histogram of APC QT intervals');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB beat QT interval
if find(a_type == 'L')
QTL_sample = (awaves(TLF) - awaves(QL)) ./ 360;
subplot(snum,1,4);
% QT_sample_i=.1:.025:1.6;
% bar(QT_sample_i,histc(QTL_sample,QT_sample_i),'hist');
hist(QTL_sample);
grid on;
title('Histogram of LBBB QT intervals');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB beat QT interval
if find(a_type == 'R')
QTR_sample = (awaves(TRF) - awaves(QR)) ./ 360;
subplot(snum,1,5);
% QT_sample_i=.1:.025:1.6;
% bar(QT_sample_i,histc(QTR_sample,QT_sample_i),'hist');
hist(QTR_sample);
grid on;
title('Histogram of RBBB QT intervals');
ylabel('# of bins');
xlabel('seconds');
end

% Paced beat QT interval
if find(a_type == '/')
QTP_sample = (awaves(TPF) - awaves(QP)) ./ 360;
subplot(snum,1,6);
% QT_sample_i=.1:.025:1.6;
% bar(QT_sample_i,histc(QTP_sample,QT_sample_i),'hist');
hist(QTP_sample);
grid on;
title('Histogram of Paced QT intervals');
ylabel('# of bins');
xlabel('seconds');
end

%% ST segment (beginning of the S wave to the beginning of the T wave)

figure(num); num = num + 1;

% Normal beat ST segment
if find(a_type == 'N')
STN_seg = (awaves(TNO) - awaves(SN)) ./ 360;
subplot(snum,1,1);
% ST_seg_i=0:.08:.8;
% bar(ST_seg_i,histc(STN_seg,ST_seg_i),'hist');
hist(STN_seg);
grid on;
```

```
title('Histogram of Normal ST segments');
ylabel('# of bins');
xlabel('seconds');
end

% PVC beat ST segment
if find(a_type == 'V')
STV_seg = (awaves(TVO) - awaves(SV)) ./ 360;
subplot(snum,1,2);
% bar(ST_seg_i,histc(STV_seg,ST_seg_i),'hist');
hist(STV_seg);
grid on;
title('Histogram of PVC ST segments');
ylabel('# of bins');
xlabel('seconds');
end

% APC beat ST segment
if find(a_type == 'A')
STA_seg = (awaves(TAO) - awaves(SA)) ./ 360;
subplot(snum,1,3);
% bar(ST_seg_i,histc(STA_seg,ST_seg_i),'hist');
hist(STA_seg);
grid on;
title('Histogram of APC ST segments');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB beat ST segment
if find(a_type == 'L')
STL_seg = (awaves(TLO) - awaves(SL)) ./ 360;
subplot(snum,1,4);
% ST_seg_i=0:.08:.8;
% bar(ST_seg_i,histc(STL_seg,ST_seg_i),'hist');
hist(STL_seg);
grid on;
title('Histogram of LBBB ST segments');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB beat ST segment
if find(a_type == 'R')
STR_seg = (awaves(TRO) - awaves(SR)) ./ 360;
subplot(snum,1,5);
% ST_seg_i=0:.08:.8;
% bar(ST_seg_i,histc(STR_seg,ST_seg_i),'hist');
hist(STR_seg);
grid on;
title('Histogram of RBBB ST segments');
ylabel('# of bins');
xlabel('seconds');
end

% Paced beat ST segment
```

```
if find(a_type == '/')
STP_seg = (awaves(TPO) - awaves(SP)) ./ 360;
subplot(snum,1,6);
% ST_seg_i=0:.08:.8;
% bar(ST_seg_i,histc(STP_seg,ST_seg_i),'hist');
hist(STP_seg);
grid on;
title('Histogram of Paced ST segments');
ylabel('# of bins');
xlabel('seconds');
end


%% RR intervals (Subsequent RP (RRs) and Previous RP (RRp))

figure(num); num = num + 1;

% Normal RR intervals

if find(a_type == 'N')
Nprev = N - 9;
RRp_N = awaves(N(2:length(N) - 1)) - awaves(Nprev(2:length(N) - 1));
Nsubseq = N + 9;
RRs_N = awaves(Nsubseq(2:length(N) - 1)) - awaves(N(2:length(N) - 1));

RRp_N_time = RRp_N ./ 360;
RRs_N_time = RRs_N ./ 360;

ratio_RR_N = RRs_N./RRp_N;
subplot(snum,1,1);
% ratio_RR_i=.5:.2:2.5;
% bar(ratio_RR_i,histc(ratio_RR_N,ratio_RR_i),'hist');
hist(ratio_RR_N);
grid on;
title('Histogram of Normal RRs/RRp intervals');
ylabel('# of bins');
xlabel('seconds');
end

% PVC RP intervals

if find(a_type == 'V')
Vprev = V - 9;
Vsubseq = V + 9;

if length(V) == 1 % If there is one PVC
    RRp_V =  awaves(V) - awaves(Vprev);
    RRs_V = awaves(Vsubseq) - awaves(V);
else
    RRp_V = awaves(V(2:length(V) - 1)) - awaves(Vprev(2:length(V) - 1));
    RRs_V = awaves(Vsubseq(2:length(V) - 1)) - awaves(V(2:length(V) - 1));
end

RRp_V_time = RRp_V ./ 360;
RRs_V_time = RRs_V ./ 360;
```

```
ratio_RR_V = RRs_V./RRp_V;
subplot(snum,1,2);
% bar(ratio_RR_i,histc(ratio_RR_V,ratio_RR_i),'hist');
hist(ratio_RR_V);
grid on;
title('Histogram of PVC RRs/RRp intervals');
ylabel('# of bins');
xlabel('seconds');
end

% APC RR intervals

if find(a_type == 'A')
Aprev = A - 9;
Asubseq = A + 9;

if length(A) == 1 % If there is one APC
    RRp_A =  awaves(A) - awaves(Aprev);
    RRs_A = awaves(Asubseq) - awaves(A);
else
    RRp_A = awaves(A(2:length(A) - 1)) - awaves(Aprev(2:length(A) - 1));
    RRs_A = awaves(Asubseq(2:length(A) - 1)) - awaves(A(2:length(A) - 1));
end



RRp_A_time = RRp_A ./ 360;
RRs_A_time = RRs_A ./ 360;

ratio_RR_A = RRs_A./RRp_A;
subplot(snum,1,3);
% bar(ratio_RR_i,histc(ratio_RR_A,ratio_RR_i),'hist');
hist(ratio_RR_A);
grid on;
title('Histogram of APC RRs/RRp intervals');
ylabel('# of bins');
xlabel('seconds');
end

% LBBB RR intervals

if find(a_type == 'L')
Lprev = L - 9;
RRp_L = awaves(L(2:length(L) - 1)) - awaves(Lprev(2:length(L) - 1));
Lsubseq = L + 9;
RRs_L = awaves(Lsubseq(2:length(L) - 1)) - awaves(L(2:length(L) - 1));

RRp_L_time = RRp_L ./ 360;
RRs_L_time = RRs_L ./ 360;

ratio_RR_L = RRs_L./RRp_L;
subplot(snum,1,4);
% ratio_RR_i=.5:.2:2.5;
% bar(ratio_RR_i,histc(ratio_RR_N,ratio_RR_i),'hist');
hist(ratio_RR_L);
```

```
grid on;
title('Histogram of LBBB RRs/RRp intervals');
ylabel('# of bins');
xlabel('seconds');
end

% RBBB RR intervals

if find(a_type == 'R')
Rprev = R - 9;
RRp_R = awaves(R(2:length(R) - 1)) - awaves(Rprev(2:length(R) - 1));
Rsubseq = R + 9;
RRs_R = awaves(Rsubseq(2:length(R) - 1)) - awaves(R(2:length(R) - 1));

RRp_R_time = RRp_R ./ 360;
RRs_R_time = RRs_R ./ 360;

ratio_RR_R = RRs_R./RRp_R;
subplot(snum,1,5);
% ratio_RR_i=.5:.2:2.5;
% bar(ratio_RR_i,histc(ratio_RR_N,ratio_RR_i),'hist');
hist(ratio_RR_R);
grid on;
title('Histogram of RBBB RRs/RRp intervals');
ylabel('# of bins');
xlabel('seconds');
end

% Paced RR intervals

if find(a_type == '/')
Rprev = P - 9;
RRp_P = awaves(P(2:length(P) - 1)) - awaves(Rprev(2:length(P) - 1));
Rsubseq = P + 9;
RRs_P = awaves(Rsubseq(2:length(P) - 1)) - awaves(P(2:length(P) - 1));

RRp_P_time = RRp_P ./ 360;
RRs_P_time = RRs_P ./ 360;

ratio_RR_P = RRs_P./RRp_P;
subplot(snum,1,6);
% ratio_RR_i=.5:.2:2.5;
% bar(ratio_RR_i,histc(ratio_RR_N,ratio_RR_i),'hist');
hist(ratio_RR_P);
grid on;
title('Histogram of Paced RRs/RRp intervals');
ylabel('# of bins');
xlabel('seconds');
end

%% Input matrix

% Normal
if find(a_type == 'N')
if find(a_type == 'N') & length(N) > 1
```

```
    QRS_N = QRS_N(2:end-1);
    PRN = PRN(2:end-1);
    PRN_seg = PRN_seg(2:end-1);
    PN_sample = PN_sample(2:end-1);
    PampN = PampN(2:end-1);
    RampN = RampN(2:end-1);
    QampN = QampN(2:end-1);
    SampN = SampN(2:end-1);
    TN_sample = TN_sample(2:end-1);
    STN_sample = STN_sample(2:end-1);
    QTN_sample = QTN_sample(2:end-1);
    STN_seg = STN_seg(2:end-1);

    inputN = [QRS_N PRN PRN_seg PN_sample PampN RampN QampN SampN...
        TN_sample STN_sample QTN_sample STN_seg ratio_RR_N RRp_N_time...
        RRs_N_time];
else
    inputN = [QRS_N PRN PRN_seg PN_sample PampN RampN QampN SampN...
        TN_sample STN_sample QTN_sample STN_seg ratio_RR_N RRp_N_time...
        RRs_N_time];
end
else
    inputN = 0;
end


% PVC
if find(a_type == 'V')
if find(a_type == 'V') & length(V) > 1
    QRS_V = QRS_V(2:end-1);
    PRV = PRV(2:end-1);
    PRV_seg = PRV_seg(2:end-1);
    PV_sample = PV_sample(2:end-1);
    PampV = PampV(2:end-1);
    RampV = RampV(2:end-1);
    QampV = QampV(2:end-1);
    SampV = SampV(2:end-1);
    TV_sample = TV_sample(2:end-1);
    STV_sample = STV_sample(2:end-1);
    QTV_sample = QTV_sample(2:end-1);
    STV_seg = STV_seg(2:end-1);

    inputV = [QRS_V PRV PRV_seg PV_sample PampV RampV QampV SampV...
        TV_sample STV_sample QTV_sample STV_seg ratio_RR_V RRp_V_time...
        RRs_V_time];
else
    inputV = [QRS_V PRV PRV_seg PV_sample PampV RampV QampV SampV...
        TV_sample STV_sample QTV_sample STV_seg ratio_RR_V RRp_V_time...
        RRs_V_time];
end
else
    inputV = 0;
end

% APC
if find(a_type == 'A')
```

```
if find(a_type == 'A')& length(A) > 1
    QRS_A = QRS_A(2:end-1);
    PRA = PRA(2:end-1);
    PRA_seg = PRA_seg(2:end-1);
    PA_sample = PA_sample(2:end-1);
    PampA = PampA(2:end-1);
    RampA = RampA(2:end-1);
    QampA = QampA(2:end-1);
    SampA = SampA(2:end-1);
    TA_sample = TA_sample(2:end-1);
    STA_sample = STA_sample(2:end-1);
    QTA_sample = QTA_sample(2:end-1);
    STA_seg = STA_seg(2:end-1);

    inputA = [QRS_A PRA PRA_seg PA_sample PampA RampA QampA SampA...
        TA_sample STA_sample QTA_sample STA_seg ratio_RR_A RRp_A_time...
        RRs_A_time];
else
    inputA = [QRS_A PRA PRA_seg PA_sample PampA RampA QampA SampA...
        TA_sample STA_sample QTA_sample STA_seg ratio_RR_A RRp_A_time...
        RRs_A_time];
end
else
    inputA = 0;
end

% LBBB
if find(a_type == 'L')
if find(a_type == 'L') & length(L) > 1
    QRS_L = QRS_L(2:end-1);
    PRL = PRL(2:end-1);
    PRL_seg = PRL_seg(2:end-1);
    PL_sample = PL_sample(2:end-1);
    PampL = PampL(2:end-1);
    RampL = RampL(2:end-1);
    QampL = QampL(2:end-1);
    SampL = SampL(2:end-1);
    TL_sample = TL_sample(2:end-1);
    STL_sample = STL_sample(2:end-1);
    QTL_sample = QTL_sample(2:end-1);
    STL_seg = STL_seg(2:end-1);

    inputL = [QRS_L PRL PRL_seg PL_sample PampL RampL QampL SampL ...
        TL_sample STL_sample QTL_sample STL_seg ratio_RR_L RRp_L_time...
        RRs_L_time];
else
    inputL = [QRS_L PRL PRL_seg PL_sample PampL RampL QampL SampL...
        TL_sample STL_sample QTL_sample STL_seg ratio_RR_L RRp_L_time...
        RRs_L_time];
end
else
    inputL = 0;
end

% RBBB
if find(a_type == 'R')
```

```
if find(a_type == 'R') & length(R) > 1
    QRS_R = QRS_R(2:end-1);
    PRR = PRR(2:end-1);
    PRR_seg = PRR_seg(2:end-1);
    PR_sample = PR_sample(2:end-1);
    PampR = PampR(2:end-1);
    RampR = RampR(2:end-1);
    QampR = QampR(2:end-1);
    SampR = SampR(2:end-1);
    TR_sample = TR_sample(2:end-1);
    STR_sample = STR_sample(2:end-1);
    QTR_sample = QTR_sample(2:end-1);
    STR_seg = STR_seg(2:end-1);

    inputR = [QRS_R PRR PRR_seg PR_sample PampR RampR QampR SampR...
        TR_sample STR_sample QTR_sample STR_seg ratio_RR_R RRp_R_time...
        RRs_R_time];
else
    inputR = [QRS_R PRR PRR_seg PR_sample PampR RampR QampR SampR...
        TR_sample STR_sample QTR_sample STR_seg ratio_RR_R RRp_R_time...
        RRs_R_time];
end
else
    inputR = 0;
end

% Paced
if find(a_type == '/')
if find(a_type == '/') & length(P) > 1
    QRS_R = QRS_R(2:end-1);
    PRP = PRP(2:end-1);
    PRP_seg = PRP_seg(2:end-1);
    PP_sample = PP_sample(2:end-1);
    PampP = PampP(2:end-1);
    RampP = RampP(2:end-1);
    QampP = QampP(2:end-1);
    SampP = SampP(2:end-1);
    TP_sample = TP_sample(2:end-1);
    STP_sample = STP_sample(2:end-1);
    QTP_sample = QTP_sample(2:end-1);
    STP_seg = STP_seg(2:end-1);

    inputP = [QRS_R PRP PRP_seg PP_sample PampP RampP QampP SampP ...
        TP_sample STP_sample QTP_sample STP_seg ratio_RR_P RRp_P_time...
        RRs_P_time];
else
    inputP = [QRS_R PRP PRP_seg PP_sample PampP RampP QampP SampP...
        TP_sample STP_sample QTP_sample STP_seg ratio_RR_P RRp_P_time...
        RRs_P_time];
end
else
    inputP = 0;
end
```

Input to ANFIS for One ECG Signal:

```
% Input data for ANFIS including both training, testing, and checking data

% Input paramters (arguments) are:
%   filename from MIT-Arrhythmia database

% Output values returned are:
%   Input training, testing, and checking data that contains input and
%   output layer for ANFIS.

% Revised: 2/7/2015 -- by Brad Funsten
%          2/23/2015 -- Added checking data for testing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
clc;
close all;
num = 1;

% Read the record ECG signal from MIT-BIH Arrhythmia Database
record = 'mitdb/101';
[sig, sampleIn, charIn,num] = WFDB_QRS(record, num);

% Obtain input layer
[inputN, inputV, inputA, inputL, inputR, inputP, num] = input_layer(sig,…
sampleIn, charIn, num);

% Output layer
outputN = ones(size(inputN,1),1);
outputV = ones(size(inputV,1),1);
outputV(1:end) = 2;
outputA = ones(size(inputA,1),1);
outputA(1:end) = 3;
outputL = ones(size(inputL,1),1);
outputL(1:end) = 4;
outputR = ones(size(inputR,1),1);
outputR(1:end) = 5;
outputP = ones(size(inputP,1),1);
outputP(1:end) = 6;

% Input to ANFIS
N_101 = [inputN outputN];
V_101 = [inputV outputV];
A_101 = [inputA outputA];
L_101 = [inputL outputL];
R_101 = [inputR outputR];
P_101 = [inputP outputP];
```

Input to ANFIS for Multiple ECG Signals:

```
% Input data for ANFIS including both training, testing, and checking data
% for multiple ECG records

% Input paramters (arguments) are:
%   filenames from MIT-Arrhythmia database associated as stored matrices

% Output values returned are:
%   Input training, testing, and checking data that contains input and
%   output layer for ANFIS.

% Revised: 2/7/2015 -- by Brad Funsten
%          2/23/2015 -- Added checking data for testing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
clc;
close all;
num = 1;
% Load Normal beats (34 of them)
 load N_100; load N_101; load N_102; load N_103; load N_104; load N_105;
 load N_106; load N_108; load N_112;
 load N_114; load N_115; load N_116; load N_117; load N_121;
 load N_122; load N_123; load N_200; load N_201; load N_202; load N_203;
 load N_205; load N_208; load N_209; load N_212; load N_213;
 load N_215; load N_220; load N_222; load N_228; load N_230; load N_231;
 load N_233; load N_234; load N_223;
% Normal = randswap([N_100; N_101; N_102; N_103; N_104; N_105; N_106;...
%     N_108; N_112; N_114; N_115; N_116; N_117; N_121; N_122; N_123;
N_200;...
%     N_201; N_202; N_203; N_205; N_208; N_209; N_212; N_213; N_215;
N_220;...
%     N_222; N_228; N_230; N_231; N_233; N_234; N_223]);

 a = randi([1 33],1);
% a = 26;
if (a==1) input_dataN=randswap(N_100); elseif (a==2)
input_dataN=randswap(N_101);
elseif (a==3) input_dataN=randswap(N_102);  elseif (a==4)
input_dataN=randswap(N_103);
elseif (a==5) input_dataN=randswap(N_104);  elseif (a==6)
input_dataN=randswap(N_105);
elseif (a==7) input_dataN=randswap(N_106);  elseif (a==8)
input_dataN=randswap(N_108);
elseif (a==9) input_dataN=randswap(N_112);  elseif (a==10)
input_dataN=randswap(N_114);
elseif (a==11) input_dataN=randswap(N_115);  elseif (a==12)
input_dataN=randswap(N_116);
elseif (a==13) input_dataN=randswap(N_117);  elseif (a==14)
input_dataN=randswap(N_121);
elseif (a==15) input_dataN=randswap(N_122);  elseif (a==16)
input_dataN=randswap(N_123);
elseif (a==17) input_dataN=randswap(N_200);  elseif (a==18)
input_dataN=randswap(N_201);
elseif (a==19) input_dataN=randswap(N_202);  elseif (a==20)
input_dataN=randswap(N_203);
```

```
elseif (a==21) input_dataN=randswap(N_205);  elseif (a==22)
input_dataN=randswap(N_208);
elseif (a==23) input_dataN=randswap(N_209);  elseif (a==24)
input_dataN=randswap(N_212);
elseif (a==25) input_dataN=randswap(N_213);  elseif (a==26)
input_dataN=randswap(N_215);
elseif (a==27) input_dataN=randswap(N_220);  elseif (a==28)
input_dataN=randswap(N_222);
elseif (a==29) input_dataN=randswap(N_228);  elseif (a==30)
input_dataN=randswap(N_230);
elseif (a==31) input_dataN=randswap(N_231);  elseif (a==32)
input_dataN=randswap(N_233);
elseif (a==33) input_dataN=randswap(N_234);
end;

% Load PVC beats (30 of them)
 load V_208; load V_106; load V_116; load V_200; load V_228; load V_105;
 load V_223; load V_233; load V_215; load V_214; load V_213; load V_203;
 load V_201; load V_100; load V_102; load V_104; load V_108; load V_111;
 load V_114; load V_118; load V_123; load V_202; load V_205; load V_209;
 load V_234; load V_107; load V_109; load V_121; load V_203; load V_230;
% PVC = randswap([V_208; V_106; V_116; V_200; V_228; V_105; V_223; V_233;...
%     V_215; V_214; V_213; V_203; V_201; V_100; V_102; V_104; V_108;
V_111;...
%     V_114; V_118; V_123; V_202; V_205; V_209; V_234; V_107; V_109;
V_121;...
%     V_203; V_230]);
b = randi([1 12],1);
% b = 6;
if (b==1) input_dataV=randswap(V_208); elseif (b==2)
input_dataV=randswap(V_106);
elseif (b==3) input_dataV=randswap(V_116);  elseif (b==4)
input_dataV=randswap(V_200);
elseif (b==5) input_dataV=randswap(V_228);  elseif (b==6)
input_dataV=randswap(V_223);
elseif (b==7) input_dataV=randswap(V_215);  elseif (b==8)
input_dataV=randswap(V_214);
elseif (b==9) input_dataV=randswap(V_213);  elseif (b==10)
input_dataV=randswap(V_203);
elseif (b==11) input_dataV=randswap(V_201);  elseif (b==12)
input_dataV=randswap(V_233);
end;
% Load APC beats (21 of them)
 load A_209; load A_222; load A_100; load A_101; load A_108;
 load A_114; load A_116; load A_117; load A_118; load A_121;
 load A_200; load A_201; load A_202; load A_205; load A_213; load A_228;
 load A_215; load A_220; load A_223; load A_231; load A_233;
% APC = randswap([A_209; A_222; A_100; A_101; A_108; A_114;...
%     A_116; A_117; A_118; A_121; A_200; A_201; A_202; A_205; A_213;
A_228;...
%     A_215; A_220; A_223; A_231; A_233]);

c = randi([1 2],1);
% c = 2;
if (c==1) input_dataA=randswap(A_209); elseif (c==2)
input_dataA=randswap(A_222);
end;
```

```
% Load LBBB beats (3 of them)
 load L_109; load L_111; load L_214;
% LBBB = randswap([L_109; L_111; L_214]);
d = randi([1 3],1);
% d = 1;
if (d==1) input_dataL=randswap(L_109); elseif (d==2)
input_dataL=randswap(L_111);
elseif (d==3) input_dataL=randswap(L_214);
end;

 % Load RBBB beats
 load R_118; load R_212; load R_231;
%  RBBB = randswap([R_118; R_212; R_231]);
 e = randi([1 3],1);
% e = 3;
if (e==1) input_dataR=randswap(R_118); elseif (e==2)
input_dataR=randswap(R_212);
elseif (e==3) input_dataR=randswap(R_231);
end

 % Load Paced beats
 load P_107; load P_102; load P_104;
% Paced = randswap([P_107; P_102; P_104]);
 f = randi([1 3],1);
% f = 3;
if (f==1) input_dataP=randswap(P_107); elseif (f==2)
input_dataP=randswap(P_102);
elseif (f==3) input_dataP=randswap(P_104);
end

%  load N2; load V2; load A2; load L2; load R2; load P2;
% load N3; load V3; load A3; load L3; load R3; load P3;

input_dataN = N_100;
input_dataV = V_208;
input_dataA = A_209;
input_dataL = L_109;
input_dataR = R_118;
input_dataP = P_107;

% input_dataN = randswap(N_101);
% input_dataV = randswap(V_233);
% input_dataA = randswap(A_222);
% input_dataL = randswap(L_111);
% input_dataR = randswap(R_212);
% input_dataP = randswap(P_102);

% input_dataN = randswap([N_100(1:100,:); N_101(1:100,:)]);
% input_dataV = randswap([V_208(1:100,:); V_233(1:100,:)]);
% input_dataA = randswap([A_209(1:100,:); A_222(1:100,:)]);
% input_dataL = randswap([L_109(1:100,:); L_111(1:100,:)]);
% input_dataR = randswap([R_118(1:100,:); R_212(1:100,:)]);
% input_dataP = randswap([P_107(1:100,:); P_102(1:100,:)]);

% Train data: 55% of total data
input_trnN = input_dataN(1:55,:); % 55% of input_dataN
```

```
input_trnV = input_dataV(1:55,:); % 55% of input_dataV
input_trnA = input_dataA(1:55,:); % 55% of input_dataA
input_trnL = input_dataL(1:55,:); % 55% of input_dataL
input_trnR = input_dataR(1:55,:); % 55% of input_dataR
input_trnP = input_dataP(1:55,:); % 55% of input_dataP

% input_trnN = Normal(1:538,:); % 55% of input_dataN
% input_trnV = PVC(1:538,:); % 55% of input_dataV
% input_trnA = APC(1:538,:); % 55% of input_dataA
% input_trnL = LBBB(1:538,:); % 55% of input_dataL
% input_trnR = RBBB(1:538,:); % 55% of input_dataR
% input_trnP = Paced(1:538,:); % 55% of input_dataP

% input_trnN = N_100(1:55,:);
% input_trnV = V_208(1:55,:);
% input_trnA = A_209(1:55,:);
% input_trnL = L_109(1:55,:);
% input_trnR = R_118(1:55,:);
% input_trnP = P_107(1:55,:);

input_trn = [input_trnN; input_trnV; input_trnA; input_trnL; ...
    input_trnR; input_trnP];
output_trn_all = input_trn(:,16);

% Check data: 10% of total data
input_chkN = input_dataN(56:65,:); % 10% of input_dataN
input_chkV = input_dataV(56:65,:); % 10% of input_dataV
input_chkA = input_dataA(56:65,:); % 10% of input_dataA
input_chkL = input_dataL(56:65,:); % 10% of input_dataL
input_chkR = input_dataR(56:65,:); % 10% of input_dataR
input_chkP = input_dataP(56:65,:); % 10% of input_dataP

% input_chkN = Normal(539:636,:); % 55% of input_dataN
% input_chkV = PVC(539:636,:); % 55% of input_dataV
% input_chkA = APC(539:636,:); % 55% of input_dataA
% input_chkL = LBBB(539:636,:); % 55% of input_dataL
% input_chkR = RBBB(539:636,:); % 55% of input_dataR
% input_chkP = Paced(539:636,:); % 55% of input_dataP

% input_chkN = N_100(56:65,:);
% input_chkV = V_208(56:65,:);
% input_chkA = A_209(56:65,:);
% input_chkL = L_109(56:65,:);
% input_chkR = R_118(56:65,:);
% input_chkP = P_107(56:65,:);

input_chk = [input_chkN; input_chkV; input_chkA; input_chkL; input_chkR;...
    input_chkP];
output_chk_all = input_chk(:,16);

% Test data: 35% of total data
input_testN = input_dataN(66:100,:); % 35% of input_dataN
input_testV = input_dataV(66:100,:); % 35% of input_dataV
input_testA = input_dataA(66:100,:); % 35% of input_dataA
input_testL = input_dataL(66:100,:); % 35% of input_dataL
input_testR = input_dataR(66:100,:); % 35% of input_dataR
input_testP = input_dataP(66:100,:); % 35% of input_dataP
```

```
% input_testN = Normal(637:978,:); % 55% of input_dataN
% input_testV = PVC(637:978,:); % 55% of input_dataV
% input_testA = APC(637:978,:); % 55% of input_dataA
% input_testL = LBBB(637:978,:); % 55% of input_dataL
% input_testR= RBBB(637:978,:); % 55% of input_dataR
% input_testP = Paced(637:978,:); % 55% of input_dataP

% input_testN = N_100(66:100,:);
% input_testV = V_208(66:100,:);
% input_testA = A_209(66:100,:);
% input_testL = L_109(66:100,:);
% input_testR = R_118(66:100,:);
% input_testP = P_107(66:100,:);

input_test = [input_testN; input_testV; input_testA; input_testL; ...
    input_testR; input_testP];
output_test_all = input_test(:,16);

% Total Data
input_data = [input_trnN ; input_chkN; input_testN; input_trnV; ...
    input_chkV; input_testV; input_trnA; input_chkA; input_testA; ...
    input_trnL; input_chkL; input_testL; input_trnR; input_chkR; ...
    input_testR; input_trnP; input_chkP; input_testP];
output_data = input_data(:,16);

% Choose input features
input_data = [input_trnN ; input_trnV; input_trnA; input_trnL; ...
    input_trnR; input_trnP; input_chkN ; input_chkV; input_chkA;...
    input_chkL; input_chkR; input_chkP; input_testN ; input_testV;...
    input_testA; input_testL; input_testR; input_testP];
input_data = input_data(:,[1,2,6,12,13,14,15,16]);
input_trn = input_trn(:,[1,2,6,12,13,14,15,16]);
input_chk = input_chk(:,[1,2,6,12,13,14,15,16]);
input_test = input_test(:,[1,2,6,12,13,14,15,16]);


% % For NN toolbox: Remove the target column from the training matrix
% input_data(:,8) = []; % overall data
% input_trn(:,8) = []; % training
% input_chk(:,8) = []; % validation
% input_test(:,8) = []; % testing
%
% % NN: output train
% outputN_on = ones(length(input_trnN),1);
% outputN_off = zeros((length(input_trnV)+length(input_trnA)+...
%     length(input_trnL)+length(input_trnR)+length(input_trnP)),1);
% outputN = [outputN_on; outputN_off];
%
% outputV_on = ones(length(input_trnV),1);
% outputVN = zeros(length(input_trnN),1);
% outputV_off = zeros((length(input_trnA)+length(input_trnL)+...
%     length(input_trnR)+length(input_trnP)),1);
% outputV = [outputVN; outputV_on; outputV_off];
%
% outputA_on = ones(length(input_trnA),1);
% outputANV = zeros(length(input_trnN)+length(input_trnV),1);
```

```
% outputA_off = zeros(length(input_trnL)+length(input_trnR)+...
%     length(input_trnP),1);
% outputA = [outputANV; outputA_on; outputA_off];
%
% outputL_on = ones(length(input_trnL),1);
% outputLNVA = zeros(length(input_trnN)+length(input_trnV)+...
%     length(input_trnA),1);
% outputL_off = zeros(length(input_trnR)+length(input_trnP),1);
% outputL = [outputLNVA; outputL_on; outputL_off];
%
% outputR_on = ones(length(input_trnR),1);
% outputRNVAL = zeros(length(input_trnN)+length(input_trnV)+...
%     length(input_trnA)+length(input_trnL),1);
% outputR_off = zeros(length(input_trnP),1);
% outputR = [outputRNVAL; outputR_on; outputR_off];
%
% outputP_on = ones(length(input_trnP),1);
% outputPNVALR = zeros(length(input_trnN)+length(input_trnV)+...
%     length(input_trnA)+length(input_trnL)+length(input_trnR),1);
% outputP = [outputPNVALR; outputR_on];
% output_trn = [outputN outputV outputA outputL outputR outputP];
%
% % NN: output check
% outputN_on = ones(size(input_chkN),1);
% outputN_off = zeros(size(input_chkV)+size(input_chkA)+...
%     size(input_chkL)+size(input_chkR)+size(input_chkP),1);
% outputN = [outputN_on; outputN_off];
%
% outputV_on = ones(size(input_chkV),1);
% outputVN = zeros(size(input_chkN),1);
% outputV_off = zeros((size(input_chkA)+size(input_chkL)+...
%     size(input_chkR)+size(input_chkP)),1);
% outputV = [outputVN; outputV_on; outputV_off];
%
% outputA_on = ones(size(input_chkA),1);
% outputANV = zeros(size(input_chkN)+size(input_chkV),1);
% outputA_off = zeros(size(input_chkL)+size(input_chkR)+size(input_chkP),1);
% outputA = [outputANV; outputA_on; outputA_off];
%
% outputL_on = ones(size(input_chkL),1);
% outputLNVA = zeros(size(input_chkN)+size(input_chkV)+size(input_chkA),1);
% outputL_off = zeros(size(input_chkR)+size(input_chkP),1);
% outputL = [outputLNVA; outputL_on; outputL_off];
%
% outputR_on = ones(size(input_chkR),1);
% outputRNVAL = zeros(size(input_chkN)+size(input_chkV)+size(input_chkA)+...
%     size(input_chkL),1);
% outputR_off = zeros(size(input_chkP),1);
% outputR = [outputRNVAL; outputR_on; outputR_off];
%
% outputP_on = ones(size(input_chkP),1);
% outputPNVALR = zeros(size(input_chkN)+size(input_chkV)+size(input_chkA)+...
%     size(input_chkL)+size(input_chkR),1);
% outputP = [outputPNVALR; outputR_on];
% output_chk = [outputN outputV outputA outputL outputR outputP];
%
% % NN: output test
```

```
% outputN_on = ones(length(input_testN),1);
% outputN_off = zeros((length(input_testV)+length(input_testA)+...
%     length(input_testL)+length(input_testR)+length(input_testP)),1);
% outputN = [outputN_on; outputN_off];
%
% outputV_on = ones(length(input_testV),1);
% outputVN = zeros(length(input_testN),1);
% outputV_off = zeros((length(input_testA)+length(input_testL)+...
%     length(input_testR)+length(input_testP)),1);
% outputV = [outputVN; outputV_on; outputV_off];
%
% outputA_on = ones(length(input_testA),1);
% outputANV = zeros(length(input_testN)+length(input_testV),1);
% outputA_off = zeros(length(input_testL)+length(input_testR)+...
%     length(input_testP),1);
% outputA = [outputANV; outputA_on; outputA_off];
%
% outputL_on = ones(length(input_testL),1);
% outputLNVA = zeros(length(input_testN)+length(input_testV)+...
%     length(input_testA),1);
% outputL_off = zeros(length(input_testR)+length(input_testP),1);
% outputL = [outputLNVA; outputL_on; outputL_off];
%
% outputR_on = ones(length(input_testR),1);
% outputRNVAL = zeros(length(input_testN)+length(input_testV)+...
%     length(input_testA)+length(input_testL),1);
% outputR_off = zeros(length(input_testP),1);
% outputR = [outputRNVAL; outputR_on; outputR_off];
%
% outputP_on = ones(length(input_testP),1);
% outputPNVALR = zeros(length(input_testN)+length(input_testV)+...
%     length(input_testA)+length(input_testL)+length(input_testR),1);
% outputP = [outputPNVALR; outputR_on];
% output_test = [outputN outputV outputA outputL outputR outputP];
%
% output_data = [output_trn; output_chk; output_test];

% % Method 2: Run ANFIS 6 times to classify heartbeats

% Normal beat vs. abnormal beat
N_trn = [input_trnN]; N_chk = [input_chkN]; N_test = [input_testN];
N_trn(:,16) = 1; N_chk(:,16) = 1; N_test(:,16) = 1;
abnormal_trn = [input_trnV; input_trnA; input_trnL; input_trnR; ...
    input_trnP]; abnormal_chk = [input_chkV; input_chkA; input_chkL;...
    input_chkR; input_chkP]; abnormal_test = [input_testV; input_testA;...
    input_testL; input_testR; input_testP];
abnormal_trn(:,16) = 0; abnormal_chk(:,16) = 0; abnormal_test(:,16) = 0;
input_trn = [N_trn; abnormal_trn];
input_chk = [N_chk; abnormal_chk];
input_test = [N_test; abnormal_test];
output_trn = input_trn(:,16);
output_chk = input_chk(:,16);
output_test = input_test(:,16);
input_chk = input_chk(:,[1,2,6,12,13,14,15,16]);
input_trn = input_trn(:,[1,2,6,12,13,14,15,16]);
input_test = input_test(:,[1,2,6,12,13,14,15,16]);
```

```
input_trn_all = [input_trnN; input_trnV; input_trnA; input_trnL;...
    input_trnR; input_trnP];
output_trn_all = input_trn_all(:,16);

input_chk_all = [input_chkN; input_chkV; input_chkA; input_chkL;...
    input_chkR; input_chkP];
output_chk_all = input_chk_all(:,16);

input_test_all = [input_testN; input_testV; input_testA; input_testL; ...
    input_testR; input_testP];
output_test_all = input_test_all(:,16);
output_data = [input_trn(:,8); input_chk(:,8); input_test(:,8)];

% % PVC beat vs. abnormal beat
% V_trn = [input_trnV]; V_chk = [input_chkV]; V_test = [input_testV];
% V_trn(:,16) = 1; V_chk(:,16) = 1; V_test(:,16) = 1;
% abnormal_trn = [input_trnN; input_trnA; input_trnL; input_trnR;...
%     input_trnP]; abnormal_chk = [input_chkN; input_chkA; input_chkL; ...
%     input_chkR; input_chkP]; abnormal_test = [input_testN; input_testA;...
%     input_testL; input_testR; input_testP];
% abnormal_trn(:,16) = 0; abnormal_chk(:,16) = 0; abnormal_test(:,16) = 0;
% input_trn = [V_trn; abnormal_trn];
% input_chk = [V_chk; abnormal_chk];
% input_test = [V_test; abnormal_test];
% output_trn = input_trn(:,16);
% output_chk = input_chk(:,16);
% output_test = input_test(:,16);
% input_chk = input_chk(:,[1,2,6,12,13,14,15,16]);
% input_trn = input_trn(:,[1,2,6,12,13,14,15,16]);
% input_test = input_test(:,[1,2,6,12,13,14,15,16]);
%
% input_trn_all = [input_trnV; input_trnN; input_trnA; input_trnL;...
%     input_trnR; input_trnP];
% output_trn_all = input_trn_all(:,16);
%
% input_chk_all = [input_chkV; input_chkN; input_chkA; input_chkL;...
%     input_chkR; input_chkP];
% output_chk_all = input_chk_all(:,16);
%
% input_test_all = [input_testV; input_testN; input_testA; input_testL;...
%     input_testR; input_testP];
% output_test_all = input_test_all(:,16);
% output_data = [input_trn(:,8); input_chk(:,8); input_test(:,8)];

% % APC beat vs. abnormal beat
% A_trn = [input_trnA]; A_chk = [input_chkA]; A_test = [input_testA];
% A_trn(:,16) = 1; A_chk(:,16) = 1; A_test(:,16) = 1;
% abnormal_trn = [input_trnN; input_trnV; input_trnL; input_trnR;...
%     input_trnP]; abnormal_chk = [input_chkN; input_chkV; input_chkL;...
%     input_chkR; input_chkP]; abnormal_test = [input_testN; input_testV;...
%     input_testL; input_testR; input_testP];
% abnormal_trn(:,16) = 0; abnormal_chk(:,16) = 0; abnormal_test(:,16) = 0;
% input_trn = [A_trn; abnormal_trn];
% input_chk = [A_chk; abnormal_chk];
% input_test = [A_test; abnormal_test];
% output_trn = input_trn(:,16);
% output_chk = input_chk(:,16);
```

```
% output_test = input_test(:,16);
% input_chk = input_chk(:,[1,2,6,12,13,14,15,16]);
% input_trn = input_trn(:,[1,2,6,12,13,14,15,16]);
% input_test = input_test(:,[1,2,6,12,13,14,15,16]);
%
% input_trn_all = [input_trnA; input_trnN; input_trnV; input_trnL;...
%     input_trnR; input_trnP];
% output_trn_all = input_trn_all(:,16);
%
% input_chk_all = [input_chkA; input_chkN; input_chkV; input_chkL;...
%     input_chkR; input_chkP];
% output_chk_all = input_chk_all(:,16);
%
% input_test_all = [input_testA; input_testN; input_testV; input_testL;...
%     input_testR; input_testP];
% output_test_all = input_test_all(:,16);
% output_data = [input_trn(:,8); input_chk(:,8); input_test(:,8)];


% % LBBB beat vs. abnormal beat
% L_trn = [input_trnL]; L_chk = [input_chkL]; L_test = [input_testL];
% L_trn(:,16) = 1; L_chk(:,16) = 1; L_test(:,16) = 1;
% abnormal_trn = [input_trnN; input_trnV; input_trnA; input_trnR; ...
%     input_trnP]; abnormal_chk = [input_chkN; input_chkV; input_chkA;...
%     input_chkR; input_chkP]; abnormal_test = [input_testN; input_testV;...
%     input_testA; input_testR; input_testP];
% abnormal_trn(:,16) = 0; abnormal_chk(:,16) = 0; abnormal_test(:,16) = 0;
% input_trn = [L_trn; abnormal_trn];
% input_chk = [L_chk; abnormal_chk];
% input_test = [L_test; abnormal_test];
% output_trn = input_trn(:,16);
% output_chk = input_chk(:,16);
% output_test = input_test(:,16);
% input_chk = input_chk(:,[1,2,6,12,13,14,15,16]);
% input_trn = input_trn(:,[1,2,6,12,13,14,15,16]);
% input_test = input_test(:,[1,2,6,12,13,14,15,16]);
%
% input_trn_all = [input_trnL; input_trnN; input_trnV; input_trnA; ...
%     input_trnR; input_trnP];
% output_trn_all = input_trn_all(:,16);
%
% input_chk_all = [input_chkL; input_chkN; input_chkV; input_chkA; ...
%     input_chkR; input_chkP];
% output_chk_all = input_chk_all(:,16);
%
% input_test_all = [input_testL; input_testN; input_testV; input_testA;...
%     input_testR; input_testP];
% output_test_all = input_test_all(:,16);
% output_data = [input_trn(:,8); input_chk(:,8); input_test(:,8)];

% % RBBB beat vs. abnormal beat
% R_trn = [input_trnR]; R_chk = [input_chkR]; R_test = [input_testR];
% R_trn(:,16) = 1; R_chk(:,16) = 1; R_test(:,16) = 1;
% abnormal_trn = [input_trnN; input_trnV; input_trnA; input_trnL;...
%     input_trnP]; abnormal_chk = [input_chkN; input_chkV; input_chkA;...
%     input_chkL; input_chkP]; abnormal_test = [input_testN; input_testV;...
%     input_testA; input_testL; input_testP];
```

```
% abnormal_trn(:,16) = 0; abnormal_chk(:,16) = 0; abnormal_test(:,16) = 0;
% input_trn = [R_trn; abnormal_trn];
% input_chk = [R_chk; abnormal_chk];
% input_test = [R_test; abnormal_test];
% output_trn = input_trn(:,16);
% output_chk = input_chk(:,16);
% output_test = input_test(:,16);
% input_chk = input_chk(:,[1,2,6,12,13,14,15,16]);
% input_trn = input_trn(:,[1,2,6,12,13,14,15,16]);
% input_test = input_test(:,[1,2,6,12,13,14,15,16]);
%
% input_trn_all = [input_trnR; input_trnN; input_trnV; input_trnA;...
%     input_trnL; input_trnP];
% output_trn_all = input_trn_all(:,16);
%
% input_chk_all = [input_chkR; input_chkN; input_chkV; input_chkA;...
%    input_chkL; input_chkP];
% output_chk_all = input_chk_all(:,16);
%
% input_test_all = [input_testR; input_testN; input_testV; input_testA; ...
%     input_testL; input_testP];
% output_test_all = input_test_all(:,16);
% output_data = [input_trn(:,8); input_chk(:,8); input_test(:,8)];

% % Paced beat vs. abnormal beat
% P_trn = [input_trnP]; P_chk = [input_chkP]; P_test = [input_testP];
% P_trn(:,16) = 1; P_chk(:,16) = 1; P_test(:,16) = 1;
% abnormal_trn = [input_trnN; input_trnV; input_trnA; input_trnL; ...
%     input_trnR]; abnormal_chk = [input_chkN; input_chkV; input_chkA;...
%     input_chkL; input_chkR]; abnormal_test = [input_testN; input_testV;...
%     input_testA; input_testL; input_testR];
% abnormal_trn(:,16) = 0; abnormal_chk(:,16) = 0; abnormal_test(:,16) = 0;
% input_trn = [P_trn; abnormal_trn];
% input_chk = [P_chk; abnormal_chk];
% input_test = [P_test; abnormal_test];
% output_trn = input_trn(:,16);
% output_chk = input_chk(:,16);
% output_test = input_test(:,16);
% input_chk = input_chk(:,[1,2,6,12,13,14,15,16]);
% input_trn = input_trn(:,[1,2,6,12,13,14,15,16]);
% input_test = input_test(:,[1,2,6,12,13,14,15,16]);
%
% input_trn_all = [input_trnP; input_trnN; input_trnV; input_trnA;...
%     input_trnL; input_trnR];
% output_trn_all = input_trn_all(:,16);
%
% input_chk_all = [input_chkP; input_chkN; input_chkV; input_chkA; ...
%     input_chkL; input_chkR];
% output_chk_all = input_chk_all(:,16);
%
% input_test_all = [input_testP; input_testN; input_testV; input_testA;...
%     input_testL; input_testR];
% output_test_all = input_test_all(:,16);
% output_data = [input_trn(:,8); input_chk(:,8); input_test(:,8)];
```

ANFIS (grid partitioning or subtractive clustering) for ECG Signals and Evaluation of Test Data:

```
% This script is the Adaptive Nuero-Fuzzy Inference System for the ECG
% classification.
%
% Input paramters (arguments) are:
%    Input and checking data from input ANFIS script.

% Output results:
%    Classified results of ANFIS for two beat types: a specific heartbeat
%    and every heartbeat type without the specific heartbeat type.

% Revised: 11/15/14 - by Brad Funsten using record 100
%           2/8/15 - More user friendly and coded ANFIS with confusion
%                    matrix for multiple records
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
% clear all;
close all;
num = 15;

% Initialize ANFIS
numMFs = 2;

% mfType = 'gbellmf'; % Gaussian bell MF
mfType = 'trimf'; % Triangular MF
% mfType = 'trapmf'; % Trapezoidal MF
% mfType = 'gaussmf'; % Gaussian curve MF
%mfType = 'gauss2mf'; % Gaussian combination MF
%mfType = 'pimf'; % Pi MF
%mfType = 'dsigmf'; % Difference between 2 Sigmoid MFs
%mfType = 'psigmf'; % Product of 2 Sigmoid MFs

strMFTYPE = str2mat(mfType);

% Grid Partition under specified MF and number
% in_fis = genfis1(input_trn,numMFs,mfType); % default is linear output MFs

% Subclustering under Gaussian MF with radii (range of influence) r
r = 0.45;
in_fis = genfis2(input_trn(:,1:7),input_trn(:,8),r,[],[1.25 0.5 0.15 1]);

% Fuzzy C means clustering under Gaussian MF with clusters k
k = 3;
% in_fis = genfis3(input_trn(:,1:7),input_trn(:,8),'sugeno',k);
%%
% Plot initial membership functions
figure(num); num = num + 1;

NumInput = size(input_trn, 2) - 1;
for i = 1:NumInput;
    subplot(NumInput, 1, i);
    plot_mf_new(in_fis, 'input', i);
    %plotmf(in_fis, 'input', i);
    xlabel(['input ' num2str(i)]);
```

```
    y = strcat('\mu','(z)');
    ylabel(y);
    set(get(gca,'YLabel'),'Rotation',0);
end
subplot(7,1,1);
title('Adapted ANFIS Membership Functions');
%%
% Training
epoch_n = 100;
[trnFIS, trnErr, ss, chkFIS, chkErr] = anfis(input_trn,in_fis,[epoch_n NaN
0.001 0.9 1.1],1,input_chk);

% Plot output training FIS membership functions
figure(num); num = num + 1;

NumInput = size(input_trn, 2) - 1;
for i = 1:NumInput;
    subplot(NumInput, 1, i);
    plotmf(trnFIS, 'input', i);
    title('Membership Functions');
    xlabel(['input ' num2str(i) ' (' strMFTYPE ')']);
end

%Plot checking FIS membership functions
figure(num); num = num + 1;

NumInput = size(input_chk, 2) - 1;
for i = 1:NumInput;
    subplot(NumInput, 1, i);
    plotmf(chkFIS, 'input', i);
    title('Membership Functions');
    xlabel(['input ' num2str(i) ' (' strMFTYPE ')']);
end

% Plot step size vs. epochs
figure(num); num = num + 1;
plot(1:epoch_n, ss);
xlabel('Iterations');
ylabel('Step size');
title('Step Curve');
grid on;

% Plot RMSE for both training and checking data vs. epochs
figure(num); num = num + 1;
%plot(1:epoch_n, trnErr,'bo','linewidth',2);
plot(1:epoch_n, trnErr*100,'linewidth',2);
xlabel('Iterations');
ylabel('RMSE (%)');
title('Error curves');
grid on;
hold on;
plot(1:epoch_n, chkErr*100,'g--','linewidth',2);
legend('Training Error','Checking Error');
%  ylim([-20 180]);
```

```
% Plot RMSE for only training
figure(num); num = num + 1;
%plot(1:epoch_n, trnErr,'bo','linewidth',2);
plot(1:epoch_n, trnErr*100,'linewidth',2);
xlabel('Iterations');
ylabel('RMSE (%)');
title('Error curve');
grid on;

% Evaluate ANFIS
trnOut = evalfis(input_trn(:,1:size(input_trn,2) - 1),chkFIS);
chkOut = evalfis(input_chk(:,1:size(input_chk,2) - 1),chkFIS);
testOut = evalfis(input_test(:,1:size(input_test,2) - 1),chkFIS);

% Training classified
figure(num); num = num + 1;
plot(1:size(input_trn,1), input_trn(:,size(input_trn,2)),'o');
hold on;
plot(1:size(input_trn,1), trnOut,'rx');
grid on;
xlabel('Heart beats');
ylabel('Classification of heart beat');
title('Training Evaluation of ANFIS');
legend('Acutal values','Trained data tested');

% Checking classified
figure(num); num = num + 1;
plot(1:size(input_chk,1), input_chk(:,size(input_chk,2)),'o');
hold on;
plot(1:size(input_chk,1), chkOut,'rx');
grid on;
xlabel('Heart beats');
ylabel('Classification of heart beat');
title('Checking Evaluation of ANFIS');
legend('Acutal values','Checking data tested');

% Testing classified
figure(num); num = num + 1;
plot(1:size(input_test,1), input_test(:,size(input_test,2)),'o');
hold on;
plot(1:size(input_test,1), testOut,'rx');
grid on;
xlabel('Heart beats');
ylabel('Classification of heart beat');
title('Testing Evaluation of ANFIS');
legend('Acutal values','Testing data tested');

% Threshold training
outThresh_Training = zeros(length(trnOut),1);
for i = 1 : length(trnOut)
    if trnOut(i) < 0.5
        outThresh_Training(i) = 0;
    elseif trnOut(i) >= 0.5
        outThresh_Training(i) = 1;
    end
end
```

```
% Threshold checking
outThresh_Checking = zeros(length(chkOut),1);
for i = 1 : length(chkOut)
    if chkOut(i) < 0.5
        outThresh_Checking(i) = 0;
    elseif chkOut(i) >= 0.5
        outThresh_Checking(i) = 1;
    end
end

% Threshold testing
outThresh_Testing = zeros(length(testOut),1);
for i = 1 : length(testOut)
    if testOut(i) < 0.5
        outThresh_Testing(i) = 0;
    elseif testOut(i) >= 0.5
        outThresh_Testing(i) = 1;
    end
end

beatClassified = 0;
for i = 1 : 35
    if outThresh_Testing(i) == 1;
        beatClassified = beatClassified + 1;
    end
end

notBeatClassified = 0;
for i = 36 : 210
    if outThresh_Testing(i) == 0;
        notBeatClassified = notBeatClassified + 1;
    end
end

% Training results
figure(num); num = num + 1;
plot(1:size(input_trn,1), input_trn(:,size(input_trn,2)),'o');
hold on;
plot(1:size(input_trn,1), outThresh_Training,'rx');
grid on;
xlabel('Heart beats');
ylabel('Classification of heart beat');
title('Training ANFIS Results After Thresholding');
legend('Acutal values','Trained data tested');
ylim([-1 2]);

% Checking results
figure(num); num = num + 1;
plot(1:size(input_chk,1), input_chk(:,size(input_chk,2)),'o');
hold on;
plot(1:size(input_chk,1), outThresh_Checking,'rx');
grid on;
xlabel('Heart beats');
ylabel('Classification of heart beat');
title('Checking ANFIS Results After Thresholding');
```

```
legend('Acutal values','Checking data tested');
ylim([-1 2]);

% Testing results
figure(num); num = num + 1;
plot(1:size(input_test,1), input_test(:,size(input_test,2)),'o');
hold on;
plot(1:size(input_test,1), outThresh_Testing,'rx');
grid on;
xlabel('Heart beats');
ylabel('Classification of heart beat');
title('Testing ANFIS Results After Thresholding');
legend('Acutal values','Testing data tested');
ylim([-1 2]);

% % Training confusion matrix
% [CTrain, order1] = confusionmat(output_trn, outThresh_Training);
% abnormalClass_Training = CTrain(1,1)./size(abnormal_trn,1).*100
% beatClass_Training = CTrain(2,2)./size(P_trn,1).*100
%
% % Checking confusion matrix
% [CCheck, order2] = confusionmat(output_chk, outThresh_Checking);
% abnormalClass_Checking = CCheck(1,1)./size(abnormal_chk,1).*100
% beatClass_Checking = CCheck(2,2)./size(P_chk,1).*100
%
% % Testing confusion matrix
% [CTest, order3] = confusionmat(output_test, outThresh_Testing);
% abnormalClass_Testing = CTest(1,1)./size(abnormal_test,1).*100
% beatClass_Testing = CTest(2,2)./size(P_test,1).*100



% surfview(chkFIS);

% Evaluate initial FIS
fuzout = evalfis(input_trn(:,1:7),in_fis);
chkfuzout = evalfis(input_chk(:,1:7),in_fis);
trnRMSE1 = norm(fuzout - input_trn(:,8))/sqrt(length(fuzout))*100
chkRMSE1 = norm(chkfuzout - input_chk(:,8))/sqrt(length(chkfuzout))*100

% Min ANFIS trn and chk error
min_chkErr = min(chkErr);
min_ckkRMSE = min_chkErr*100
iteration_chkErr = find(chkErr == min_chkErr)
min_trnErr = trnErr(iteration_chkErr)*100

beatClassified
notBeatClassified
```

Comparison with ANN (Gradient Descent and Levenberg-Marquardt):

```
% This script compares the ANFIS with ANN under three training algorithms:
% Gradient Descent, Levenberg-Marquardt, and Radial Basis Networks.
%
% Input paramters (arguments) are:
%    Input training, testing, and checking data with target data as one-hot
%    encoding scheme.

% Output results:
%    Classified results of ANN for six heartbeat types: Normal, PVC, APC,
%    LBBB, RBBB, and Paced beats. RMSE curves for training and checking.
%    Confusion matrices for performance evaluation.

% Revised: 4/15/15 -- by Brad Funsten
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
close all;
%clear all;

num = 1; % figure number

%%

% Create a Pattern Recognition Network
net = patternnet(7,'traingd');

% List of Training algorithms:
%trainlm  Levenberg-Marquardt
%trainbr  Bayesian Regularization
%trainbfg BFGS Quasi-Newton
%trainrp  Resilient Backpropagation
%trainscg Scaled Conjugate Gradient
%traincgb Conjugate Gradient with Powell/Beale Restarts
%traincgf Fletcher-Powell Conjugate Gradient
%traincgp Polak-Ribiére Conjugate Gradient
%trainoss One Step Secant
%traingdx Variable Learning Rate Gradient Descent
%traingdm Gradient Descent with Momentum
%traingd  Gradient Descent

% Divide train, validation, and test data
[trainInd,valInd,testInd] =
divideind(length(input_data),1:length(input_trn),length(input_trn)+1:length(i
nput_trn)+length(input_chk),length(input_trn) + length(input_chk) + 1:
length(input_trn)+length(input_chk)+length(input_test));
%trainInd = 1:length(input_trn);

net.divideFcn = 'divideind';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainInd = trainInd; % put in training data
net.divideParam.valInd = valInd; % put in validation data
net.divideParam.testInd = testInd; % put in testing data
net.trainParam.lr = 1; % learning rate
net.trainParam.goal=1e-7; % reach mse of 10^-7
```

```
net.trainParam.epochs = 10000; % number of iterations
net.layers{1}.transferfcn = 'tansig';  % hidden layer 1 activation function
net.layers{2}.transferFcn = 'tansig'; % output layer activation function

% Train the Network
data = input_data';
target = output_data';
% [net,tr] = train(net,data,target);
%
% % Weights and bias'
% weight_final = net.iw;
% weight_final_layer = net.lw;
% bias_final = net.b;
%
% % Put same training data through NN
% out = net(data);
% errors = gsubtract(target,out);
% performance = perform(net,target,out);
%
% % Test trained NN
% out_test = net(input_test');
%
% %figure(num); num = num + 1;
% %plotperform(tr);
% mse_tr = tr.perf(1:tr.num_epochs);
% mse_val = tr.vperf(1:tr.num_epochs);
% mse_test = tr.tperf(1:tr.num_epochs);
% rmse_tr = sqrt(mse_tr);
% rmse_val = sqrt(mse_val);
% rmse_test = sqrt(mse_test);
%
% % RMSE
% figure(num); num = num + 1;
% plot(1:tr.num_epochs, rmse_tr*100,'b','linewidth', 2);
% hold on;
% plot(1:tr.num_epochs, rmse_val*100,'r--','linewidth', 2);
% %hold on;
% %plot(1:tr.num_epochs, rmse_test*100,'r','linewidth', 2);
% legend('Training Error','Checking Error');
% %legend('Train','Validation','Test');
% title('Error curves');
% xlabel('Iterations');
% ylabel('RMSE (%)');
% grid on;
%
% % Classes assined
% classes = (vec2ind(out))';
%
% % Plots
% % Uncomment these lines to enable various plots.
% %figure, plotperform(tr);
% figure(num); num = num + 1;
% plotconfusion(output_test',out_test);
% %figure, plotroc(target,outputs);
```

APPENDIX C – Project Analysis

Project Title: ECG Classifiation with an Adaptive-Neuro Fuzzy Inference System
Student's Name: Brad Thomas Funsten

Summary of Functional Requirements
This project detects abnormalities in ECG signals using an adaptive neuro-fuzzy system. In addition, an algorithm is used to extract features in an ECG signal for the proposed classification system.

Primary Constraints
Heart rate variability, nonlinear heartbeats between patient to patient makes it difficult for an effective classificiation. Proposed system is computationally expensive in terms of inputs and updating parameters.

Economic
Human Capital – This project is mean to use minimal human capital. This project aims to reduce the need for humans in ECG classificiation.
Financial Capital – Only man-hours to deploy system is used, as no physical materials were used.
Manufactured or Real Capital – Program should be able to run on any general purpose computer.
Natural Capital – Due to the program's ability to run on any general purpose computer, natural capital should only be limited to the cost of computers already made in use.

The only costs accruing in the project lifestyle was the purchase of MATLAB ® student edition, Fuzzy Logic Toolbox, and Neural Network Toolbox ($160). Benefits accrue at the completion of the various tests for the program.

Inputs required to the system are ECG data taken by a user or doctor. This project costs $160 from beginning to end and is paid by the designer of the project.

This project earns no money. It can both help reduce cost for patient care and increase accessibility to care.

No additional maintenance or operation costs occur apart from regular computer maintenance.

Estimated Development Time: 12 months

After the project ends, future work can be done to improve results or add more features.

If manufactured on a commercial basis:
Manufacturing is anticipated to not be on a commercial basis. Higher accuracy would be necessary to be competitive in the medical industry.

Environmental
The only environmental resources needed are electricity to run a computer. This project only uses natural resources indirectly to generate power (e.g. coal, oil, and natural gas). This project affects species other than humans indirectly. The project affects humans through input data and output results of patients and users control of the program.

Manufacturability
As there is no physical component to the project (only software), manufacturability is limited to software distribution. This can be done over the internet for little cost. Issues associated with manufacturing include the creation of a link to upload the program. This would involve a log-in and registration in order

to purchase the program. A free trial for a certain number of days would be required in order to provide incentive to buy the program.

Sustainability

Manufacturing and upgrading/updating the system are limited to software distribution. The only end of life concern is indirect (computer disposal). Updating the design would be through an increase of the amount of nodes for the neural network leading to an increase in computational power. An issue in updating the project is increased consumption of electrical power through the project designer's time used in updating the project.

Ethical

Under IEEE ethical code one, the user or doctor must accept responsibility in making decisions through this program in terms of safety, health and welfare of the public. The user or doctor must not endanger the public or environment with false results from the program. The code goes along with the third code where users must be honest and realistic in stating claims or estimates based on available data (ECG results). Once sold on the internet, the implications of the program being cited or modified would have to be permissible by the designer of the project.

In terms of utilitarianism, where the standard of the greatest good for the greatest number is practiced, this program would provide a way for any user to diagnose a heart disease. This benefits the greatest number because there are more people benefited than having only doctors diagnose the heart disease.

Health and Safety

The only concern is the patient's ability to partake in an ECG test.

Social and Political

A political issue associated with the design is indirect. If this project leads to a government issue into which it is purchased by the government or restricted for some reason, then political implications are involved. This project impacts patients socially through the results. The patient's behavior is a social implication from the ECG results.

Direct stakeholders: Patients, doctors, cardiologists and users.

Indirect stakeholders: humans affected socially, politically, environmentally, and ethically.

The extent to which the stakeholders benefit equally is the both the high accuracy and the valid results of the ECG test. The user or doctor and patient do not pay equally in terms of buying the program and performing tests with it on patients. The doctor pays for one program per computer and receives income from their patients who go through the tests. This may cause inequities since the number of computers with the program must equal the number of patients being tested.

Development

A new tool used in the development that was learned independently during the course of the project was the Adaptive Neuro-Fuzzy System in the Fuzzy Logic Toolbox. A new technique that was learned independently during the course of the project was to effectively classify six type of heartbeat types and compare with several artificial neural networks.