# Bradycardia Detection and Prediction

**Faraz Khan**
Ira A. Fulton Schools of Engineering
Arizona State University
faraz.khan@asu.edu
1000287028

**Varuni Shama Rao**
Ira A. Fulton Schools of Engineering
Arizona State University
vshamara@asu.edu
1211222841

**Syed Saad Imam**
Ira A. Fulton Schools of Engineering
Arizona State University
ssimam@asu.edu
1205241072

**Bernard Ngabonziza**
Ira A. Fulton Schools of Engineering
Arizona State University
bngabonz@asu.edu
1207993380

## ABSTRACT
In this project we create a mobile app to interpret ECG data to detect and predict Bradycardia condition for the human heart, using statistical as well as Deep Machine Learning methods. Data from a wearable ECG sensor is utilized. In the first method we use the ARIMA statistical model to predict Bradycardia events, ahead of time, based on statistical variations. In the second method we used Recurrent Neural Network (RNN) Regression with Long Short-Term Memory (LTSM) to predict future Bradycardia events by converting the input ECG into time-series instances. Prediction with RNN was not successful. We also used Support Vector Machine to detect and predict Heart Rate and Bradycardia. Detection/Classification was 95% accurate, but Prediction with SVM was not attempted.

## Keywords
ECG; Bradycardia; ARIMA; Deep Learning; Recurrent Neural Networks; Detection; Forecast; Prediction; LTSM.

## INTRODUCTION
Many heart problems such as Bradycardia and Tachycardia etc. can be diagnosed from ECG signals. We investigate the task of Bradycardia detection and prediction from ECG signal captured from a wearable sensor. This is known to be a difficult task for computers but can usually be determined by an expert from a single, well-placed lead. The importance of ECG classification and prediction is very important due to many current medical applications. Currently, there are many statistical and machine statistical and learning solutions which can be used for classifying and prediction condition based on ECG data. Many of the shallow machine learning solutions require extensive training data with accurately supervised classification or annotation using well-known features found in the data. In this project, however we have attempted to use Deep Learning in which the first layer acts as a feature extractor while the hidden layer, implements the classification, decision and regression functionality, that is the characteristic of recurrent neural networks.

## PROJECT SETUP
Mobile:

- Android Emulator with at least 2GB RAM
- Android Studio 3.1 for mobile app development

Deep Learning:

- Windows 10 OS
- Deep Learning for Java (DL4J) Library
- IntelliJ IDEA for RNN coding
- NetBeans with JDK 8 for all other coding

## IMPLEMENTATION
The project has been divided into the following task. Each section lists the explanation of the task, major algorithm or method used, other implementation details, and intermediate results if applicable.

### 1. Data Collection:

ECG data was collected over a period of 5 days by each member of the team, over a period of 8 hours a day. A wearable ECG sensor called "90° eMotion Faros ECG Sensor", provided a single channel high-quality wireless ambulatory signal and recorded the data using the EDF file format. The sensor also contained a 3-Channel (X-Y-Z) Accelerometer. Both the ECG and Accelerometer recorded data at a sampling rate of 250Hz.

**Figure 1: Single Channel Faros 90° eMotion Wearable ECG Sensor and Accelerometer.**

### 2. Data Conversion:

For the purpose of this project, and for ECG simulation over time, the data was not used a stream. Rather the entire data file was converted to CSV file format with two 2 columns: Time Index and ECG Signal Value. In a real world implementation, the data should be read as a

continuous stream rather than stored files. We used this method to simulate the passage of time by skipping ahead the time index values.

### 3. Data Preprocessing

No manual data removal was done. This is to avoid manual cleanup because we wanted to treat the data as close to the real world as possible, where no manual preprocessing can be done.

The first automatic preprocessing step is to create a smooth version of the ECG signal for use in the detection of the QRS peaks. For this we used Kalman Filtering which removes the small and large fluctuations of the ECG signals. We needed a smooth baseline (similar to a moving average) signal path that follows the overall movement of the ECG signal but excludes the major peaks, as well as the noisy fluctuations of the signal.


**Figure 2: Raw ECG, Kalman Filtering and Peak Detection.**

### 4. Peak Detections

The next step in the preprocessing is the detection of the QRS event, i.e. the peaks found in the raw ECG signal. While Kalman Filtering can also be used to create approximate peaks in the signal, we used a different technique, which gave us precise location of the peaks and the time indices. Kalman Filter beside giving a lagging output also gives an approximate peak which is a smoothed version of the raw signal.

Instead we used a modified version of the Smoothed Z-Score Algorithm. "It is based on the principle of dispersion: if a new data point is a given x number of standard deviations away from some moving mean, the algorithm signals (also called z-score). The algorithm is very robust because it constructs a separate moving mean and deviation, such that extreme signal values do not corrupt the threshold. Future signals are therefore identified with approximately the same accuracy, regardless of the amount of previous signals." [2]

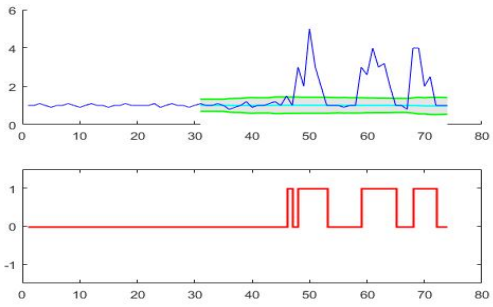

**Figure 3: Peak detection using Smoothed Z-Score**

The modification made is that the running average used in the original algorithm is replaced by Kalman Filtered values of the raw ECG signal. Kalman Filtering smooths the erratic signal much better than a moving average.

Using the Kalman values, any data points outside 2.5 times the average value were marked as potential peaks. Figure 3 shows the simulated results.

Because of the high sampling rate and the use of the Smoothed Z-Score peak detection method, multiple data points were marked as peaks. There it became necessary to remove redundant peak by combing several consecutive ones into a single well-defined peak. These peaks appear are clustered together.

### 5. Declustering Peaks

The peak detection code detects peaks in rapid succession, next to each other, because the sampling rate is too high and multiple data points are identified as potential peaks. To fix this problem, we remove peaks that are adjacent to each other and keep only 1 of them.

The average QRS peak in in a human heart lasts for less than 100ms [3] (which is equal to 25 samples given the sampling rate of 250Hz). This means that within the 100ms window there can never be more than one peak. This step goes through every 100ms (i.e. 25 consecutive samples) and remove any peaks that exists more than once. In other words, every 100ms windows can have 0 or 1 peaks, but never 2 or more. This Declustering gives a more accurate result than without it.

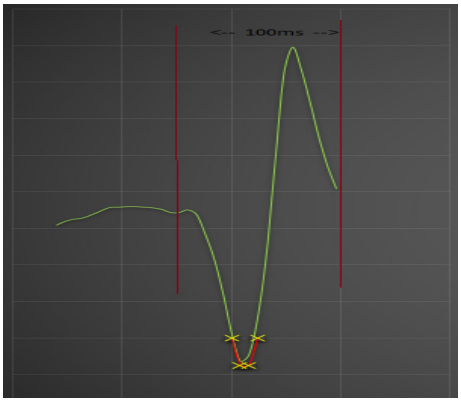

**Figure 4: Multiple data points identified as peak. Declustering removes all but one peak.**

## 6. Heart Rate Calculations

Once all the peaks have been detected and declustered, we use a simple peak counting algorithm to count the number of peaks in a 60 second sample window. This gives us the current heart rate at the given instance of time, with a resolution of one min.

The variation of the heart is plotted for reference. This variation is also used in the next phase of the project where we attempt to detect and predict the Bradycardia event.

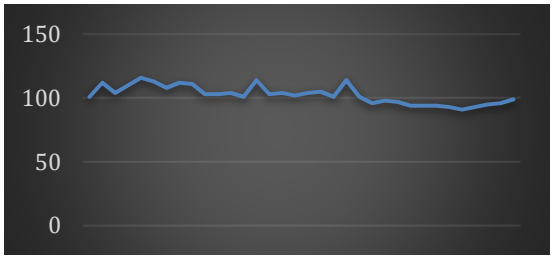


**Figure 5: Example of Heart Rate Variation over a period of 30 minutes.**

## 7. Training Files for Deep Learning

Once all the peaks have been detected, we take the ECG signal values and time indices where the peaks occur and record them into CSV files compatible with the "Deep Learning 4 Java" RNN Library. There is one file for each user who worn the sensor. The 8-hour data file is broken down into 1-min segments each with 60*250 samples. Each of the 1-min segment is saved in a CSV file with 60*250 columns and N rows where N is the number of 1-min segments.

Each row representing a segment is labeled with the heart rate calculated from within that segment. The heart rate if just the number of peak in that 60-second period. This is the class label that is used along with the segment data as the training input to the RNN.

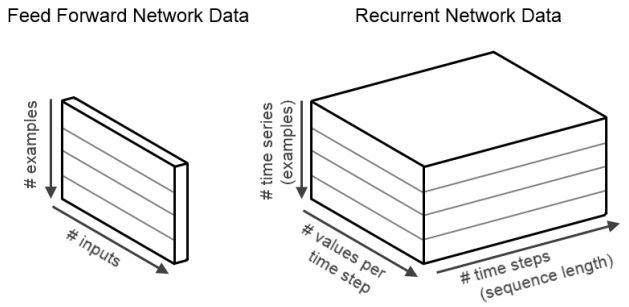


**Figure 6. Preparing CSV files for training Recurrent Neural Network vs a Feed Forward Network. [5]**

## 8. Predicting Bradycardia using ARIMA Method.

In this phase we predict the Bradycardia event (i.e. heart rate lower than 60 beats per minute) using a simple statistical method called AutoRegressive Integrated Moving Average (ARIMA) forecasting. This is a robust and widely used statistical method for time series forecasting where both local and long term fluctuations in the data is taken into account. In practice ARIMA models have been used in short term stock market price predictions, and weather forecasts. In fact, the model was originally developed for its application in weather forecasting. Some of the terms such as "Seasonal Parameters" indicate the history of the ARIMA forecasting method. ARIMA works on lagging input data and outputs a series of time forecast based on the observed structure and variation in the data. Then the model generates a series of predictions.

In this project we chose a linear model for prediction which means the (p, d, q) parameter will produce a linear prediction after a series of past data has been observed. The heart rates of the past 30 minutes are fed into the model, and the next 20 minutes are forecasted. If the forecast included a BradyCardia heart rate (i.e; < 60) then the app shows that to the user.

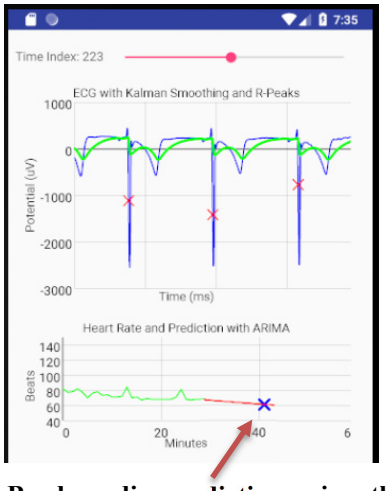


**Figure 7. Bradycardia prediction using the ARIMA Forecasting method. The green line is the past heart rate, while the red line is the projected heart rate. The blue cross is the projected Bradycardia event.**

Using this method, the projected time of Bradycardia event was between 5 and 15 minutes ahead of the occurrence. This result is not accurate because of the lack of testing real data. We only had access to one test file with a real Bradycardia event. Nevertheless, in a real world situation any advance warning can be enough to warn the user to avoid any medical complications.

Since we only work with a small set of data at a time (i.e. 30 minutes of samples) the execution time for the peak detection, kalman smoothing, delcustering and ARIMA

forecasting is quite low. We obtain the following execution times:

| Activity | Execution Time |
|---|---|
| Load 8 hrs of data | 30 seconds |
| Load 30 mins of data | 5 ms |
| Kalman Smoothing | 238ms |
| Peak Detection | 287ms |
| Declustering | 30ms |
| Heart Rate Calculation | 10ms |
| ARIMA Forecasting | 15ms |

**Table 1: Execution Times**

## 9. Predicting Bradycardia using Deep Learning with RNN and LSTM.

In this phase we use a deep learning method to predict the Bradycardia event. For this project we have chosen to use Recurring Neural Network with Long Short-Term Memory based on the Deep Learning for Java (DL4J) library.

Time series predictions using a neural network is a difficult task since the input adds the complexity of a time based sequence dependence among the input variables. A powerful type of neural network designed to handle sequence dependence is called Recurrent Neural Networks. The Long Short-Term Memory network or LSTM network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained.

The Long Short-Term Memory network, or LSTM network, is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem that most Feed Forward network suffer from. [5]

The training files were generated during the preprocessing and peak detection stages and are described in section 7.

Each line in the CSV file represents a 60-second instance of the normalized ECG data. Every R-peak signal is normalized to 1 while all other signals are normalized to 0.

The format of a training CSV file contains the class label of the data input instance in the 1st column, which is the Heart Rate (i.e. the number of peak signals in the instance. While the rest of the normalized signals follow in the columns 2 to 15,000. The file contains the entire 8 hours of data with 1 minute of data on each line.

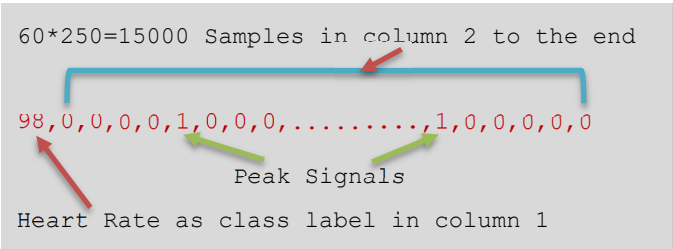


**Figure 8: Sample layout for CSV training file for RNN.**

We used the regression in RNN with LTSM, however our efforts were not successful and no useable results were obtained.

We implemented the RNN code in IntelliJ IDEA environment because the DL4J library's source code and example are designed to run in IntelliJ with dependence on the Maven Repository.

We did not implement RNN in the mobile app.

The RNN we used had one input layer, 2 hidden layers and 1 output layer. [5]

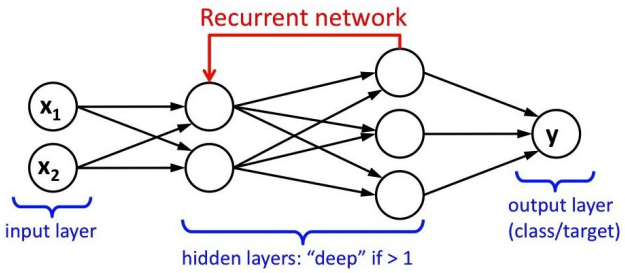


**Figure 9: RNN Input and output layers**

The results obtained were not satisfactory. The predicted values of the heart rate remained close to linear and did not provide any insight into the projected heart rate.

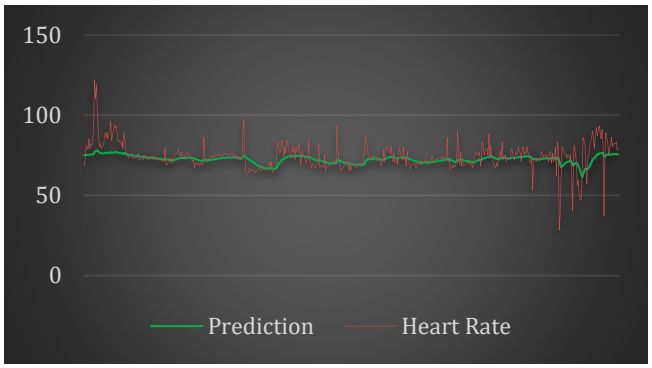


**Figure 10: RNN testing results showing training and predicted data.**

## 10. Machine Learning with SMV

We also trained an SVM machine to detect and predict Bradycardia event. Although the heart rate and Bradycardia detection was successful with this method, there was no success with advance prediction.

The training file consisted for data from over 20 volunteers. Each line in the input file consisted of a label that was the manually calculated heart rate followed by data points over a 1- min period.

We used LibSVM to train a model to calculate heart rate and detect Bradycardia condition. A 5-fold cross validation showed the following results:

Optimized Results using 5-Fold Cross validation:

Cost=2.0

Gamma=0.0078125,

Cross Validation Accurate = 95%

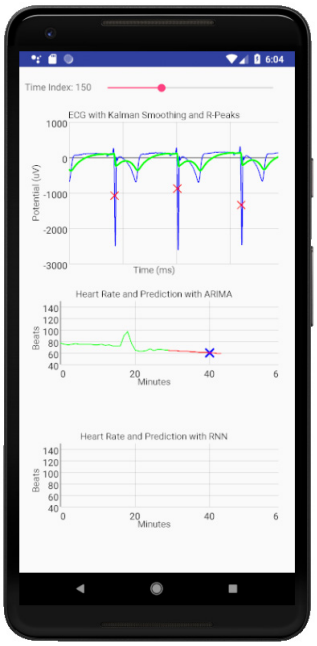


**Figure 9: App running in Simulator. The RNN and SVM implementation is not shown.**

## LIMITATIONS

The app was implemented with the simulation in mind. Therefore, the entire ECG data file was loaded into memory. This will not work in real implementation since the mobile app will have limited memory.

The training for SVM and RNN has to be done offline. Which mean the model has to be pre-trained before it is used in the app.

## CONCLUSIONS

We were able to successfully create a heart monitoring mobile app that utlized the data provided by an ECG sensor to detect and predict Bradycardia condition using 2 different methods.

The ARIMA statistical forecasting method was able to predict Bradycardia event. The result shows 5-15 min projected time of prediction. However, this result is not accurate due to lack of testing data with actual Bradycardia events.

Computation of Heart Rate and Detection of Bradycardia with SVM was successful at 95% accuracy rate with 5-fold cross validation. However, prediction with SMV was not attempted.

Prediction with a Deep Learning (RNN with LSTM) was attempted but even with much effort it was not successful. No meaningful results were obtained.

## 13. Completion of Tasks

| No. | Task | Status | Saad | Varuni | Bernard | Faraz |
|---|---|---|---|---|---|---|
| 1.1-1.4 | Collect Data | Complete | 1 | 1 | 1 | 1 |
| 1.5 | Accept | Complete | 1 | 1 | 1 | 1 |
| 2.1 | Clean Data | Complete | 1 | 1 | 1 | |
| 2.2 | Remove Artifacts | Complete | 1 | 1 | 1 | |
| 2.3.1 | Kalman | Complete | | | 1 | |
| 2.3.2 | Declustering | Complete | | 1 | 1 | |
| 2.3.3 | Peek Detect | Complete | 1 | | | |
| 2.4 | Heart Rate | Complete | 1 | | | |
| 2.5 | Plot HR | Complete | 1 | 1 | 1 | 1 |
| 3.1 | Detect BC | Complete | | | 1 | |
| 3.2-3.3 | Annotate BC | Complete | | 1 | | 1 |
| 3.4 | Mobile App | Complete | | 1 | | 1 |
| 3.5 | Execution Time | Complete | 1 | | | 1 |
| 4.1 | Plot Variance | Complete | | | | 1 |
| 4.2 | Threshold | Complete | | | | 1 |
| 4.3 | Evaluate | Complete | 1 | 1 | 1 | 1 |
| 4.5 | K-Fold | Complete | | | | 1 |
| 4.6 | RNN | Complete | | | | 1 |
| | | Total: | 9 | 9 | 9 | 11 |


## ACKNOWLEDGMENTS

We acknowledge ASU Impact lab for contributing the Kalman Filtering algorithm and providing the sensor for use in this project. Also we acknowledge Deep Learning 4 Java site for the RNN library and code, and Jean Paul, for the Smoothed Z-Score algorithm.

## REFERENCES


1. B Pyakillya et al, “Deep Learning for ECG Classification”, 2017 J. Phys.: Conf. Ser. 913 012004
2. Jean Paul, “Smoothed Z-Score Algorithm for Peak Detection”, Stack Exchange, Retrieved 4/15/2018. https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362
3. Ziad F. Issa et al., “Clinical Arrhythmology and Electrophysiology: A Companion to Braunwald's Heart Disease (Second Edition), 2012
4. Adam Gibson et al. “Recurrent Neural Networks in DL4J”, Retrieved 4/29/2018. https://deeplearning4j.org/usingrnns
5. Jason Brownlee, “Time Series Prediction with LSTM Recurrent Neural Networks”, Retrieved 4/29/2018. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/