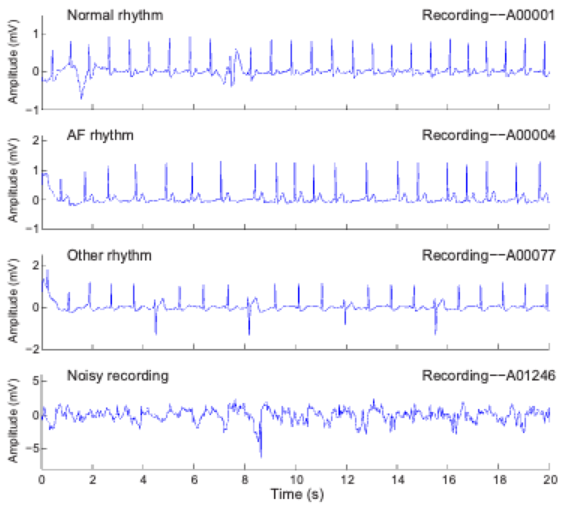
Diagnose Atrial Fribrillation from ECG data via sequence classification

1 Background

Atrial Fibrillation (AF) is a cardiac disease occurring in 1-2% of the general population. It is associated with significant mortality and morbidity through association of risk of death, stroke, hospitalization, heart failure and coronary artery disease, etc. More than 12 million Europeans and North Americans are estimated to suffer from AF, and its prevalence will likely triple in the next 30-50 years.

Electrocardiogram (ECG) are recordings of the electrical activity of the heart for short periods of time. ECGs are used to detect abnormal heart rhythms, and general physicians generally have the equipment used to record them, leading to wide availability as a front-line detection method for heart disease. ECGs normally use a total of 12 sensors (leads) placed at different locations on the body to record the electrical activity of the heart from different viewpoints.

In this project, you are given 8,528 labeled ECG readings (but only for a single lead) for patients with normal rhythms, AF, and non-AF abnormal rhythms. The recordings were taken at a frequency of 300 Hz, last from 9 to 60s, and are reported in millivolts (mV). Some of the readings are noisy due to failures in the data collection and are marked as such. Your goal is to develop a classifier that will categorize each ECG as indicating a normal, AF, non-AF abnormal, or undetermined (noisy) rhythm.



2 Instructions

1. **Download and Explore the Data**

   - Download the data from the The PhysioNet/Computing in Cardiology Challenge 2017 web site. You will need two files: the **training2017.zip** file and the **sample2017.zip** file.
   - Unzip these. Within training2017 you will find files that end in a .mat extension. These are the ECG sequence data to be use for training. You will also find REFERENCE.csv. This contains the label for each sequence. Within sample2017/validation you will also find files that end in a .mat extension and a REFERENCE.csv. These are the sequences and labels to be used for validation.
   - Use Scipy's loadmat() function to load the files, i.e. from scipy.io import loadmat. Start by loading some the recordings and plotting them. They should look like the examples in the image above. Count the total number of recordings to ensure that it matches the number given above. Generate a table and count plot for the number of recordings in each category. Look at the distribution of lengths of the recordings – are they the same?

2. **Build several deep learning architectures to classify this data.**

   - The ECG sequences are not the same length. There are model architectures available for dealing with sequences of arbitrary length, but for simplicity we will preprocess these sequences so that they have the same length. There are many possible ways to do this, for example:
     - Pad the end of each sequence with zeros to make it the length of the longest overall sequence;
     - Take the first t datapoints for each sequence, where t is the length of the shortest sequence;
     - Sample a window of each sequence of length t, where t is less than or equal to the length of the shortest sequence;
   - Choose and implement a strategy for making the sequences the same length.
   - Use Keras to build a dense NN to classify these sequences. Begin with a single hidden layer. Use accuracy as your metric and Adam as your optimizer.
   - Use Keras to build a second deep learning model, either a Convolutional neural network or a transformer, to classify these sequences. Use accuracy as your metric and Adam as your optimizer.
   - Evaluate your models on the validation set using accuracy. Also, construct confusion matrices.
   - Optimize your deep learning models by experimenting with different choices for depth and width, different regularization choices like dropout, etc.

3 Submission Instructions

Write up your analysis in a 2-page report. Your report should address:

- Background on the data set. What is the structure of this ECG data set? Are there any special considerations needed when modeling it?
- Include the summaries of your optimized models in the report. Why did you make the architecture choices that you did as you optimized them?
- What were the impact of the various parameters on model performance? Which changes increased, decreased, or did not change accuracy? What changes were the most beneficial? (Think of what advice you might give to someone who wanted to do a similar project in terms of using their time most efficiently.)
- Which model performed better? Why do you think this is?
- Interpret the confusion matrices. Were the errors balanced across the classes for both models? Were some classes harder to correctly predict than others? If not, why do you think this is?

Submit your lab report as a PDF and your code as a notebook file in canvas.