Anomaly Detection with Autoencoders & Optimization Techniques

Aaron Daniel

Table of Contents

Identification of the Problem

The topic of this report is anomaly detection in human heart ECGs. ECG, or electrocardiogram, is an electrogram of the heart which is a graph of voltage versus time of the electrical activity of the heart using electrodes placed on the skin. Heart conditions are a leading cause of death in the U.S. and worldwide, and ECGs are an essential diagnostic tool to monitoring and diagnosing heart-related health issues. Most ECG tests are done in a clinic or hospital setting (initially). But we may be nearing an age where ECG and heart activity can be monitored at low cost and high reliability with relatively little constraints through wearable technology.

The capability and availability of wearable technology (with healthcare use potential) has increased greatly in the last decade. These devices are largely affordable and accessible for many Americans and others around the world, as demonstrated by the huge success of the industry. The Fitbit emerged as a huge hit in 2009 - and nine years later, the Apple Watch Series 4 was released with capability to capture its wearer's ECG.

Some constraints associated with detecting abnormal cardiac activity with wearable technology may be data availability, power use, and reliability. The cost of both false positives and false negatives are quite high - especially when the cost of missing an abnormal cardiac event may mean greater risk of illness or death. The constraints of data availability and reliability are coupled - although one can generalize normal ECG appearances and ranges, no two humans or heartbeats will be exactly alike. However, the advent of wearable technology brings the immediate availability of personal health data to our fingertips.

One approach to analyzing ECG data can be an approach involving object detection and autoencoders. The baseline code and model used for this comes from a Tensorflow colab notebook on an introduction to autoencoders.

Implementation of solution in Tensorflow

Future wearable technology may allow for a single user's health data to be recorded and stored, so that any anomaly detection or warning systems would be tailored to that individual's health data, not a collection of data from many individuals from a diverse set of backgrounds (socioeconomic, cultural, etc.) which may render inferences from the dataset much less relevant to an individual.

For this report, the original dataset used is a 20-hour long ECG downloaded from Physionet called BIDMC Congestive Heart Failure Database, originally published in Physionet. The recording was captured from a patient with severe congestive heart failure.
This dataset was preprocessed in the following steps:

1. Extract each individual heartbeat
2. Make each heartbeat equal length using interpolation
3. Normalize the data to the range [0,1]

Furthermore, for this anomaly detection approach an autoencoder was used, only trained on the normal ECG signals - signals labeled as abnormal are only used for scoring later on.

In the baseline model used in this report, an autoencoder is employed for the task - this allows for the ECG samples to be encoded into a lower dimensional latent representation, then decoded back into a "fully detailed" representation - in other words, it compresses the data while minimizing reconstruction error. The structure is captured in the below to clips from the notebook:

```
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Dense(32, activation="relu"),
      layers.Dense(16, activation="relu"),
      layers.Dense(8, activation="relu")])

    self.decoder = tf.keras.Sequential([
      layers.Dense(16, activation="relu"),
      layers.Dense(32, activation="relu"),
      layers.Dense(140, activation="sigmoid")])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = AnomalyDetector()
```

```
Model: "anomaly_detector_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_2 (Sequential)   (None, 8)                 5176

 sequential_3 (Sequential)   (None, 140)               5308

=================================================================
Total params: 10,484
Trainable params: 10,484
Non-trainable params: 0
_________________________________________________________________
```

Fig. 1: Model Architecture

In a real world situation, a model would not typically have access to data representing “abnormal” heart activity of the individual. Therefore, this model undergoes unsupervised learning using only normal activity, but evaluated using the full test set. Training is accomplished in mere seconds over 20 epochs, using an Adam optimizer and Mean Absolute Error loss.

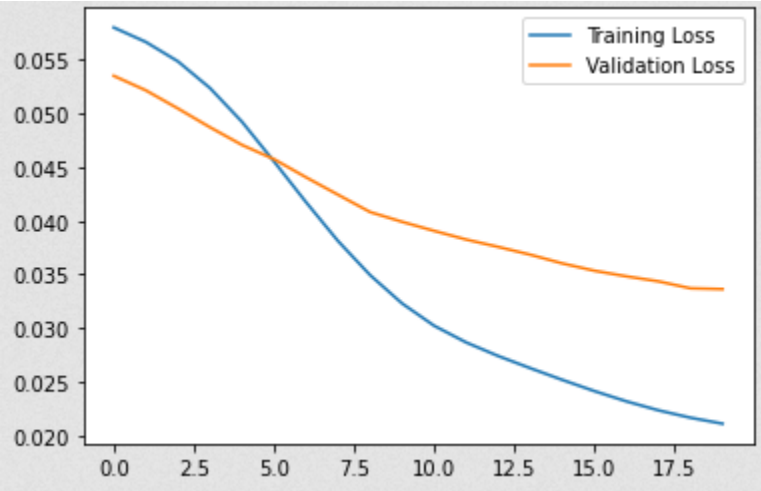

Fig. 2: Baseline Model Training and Validation Loss

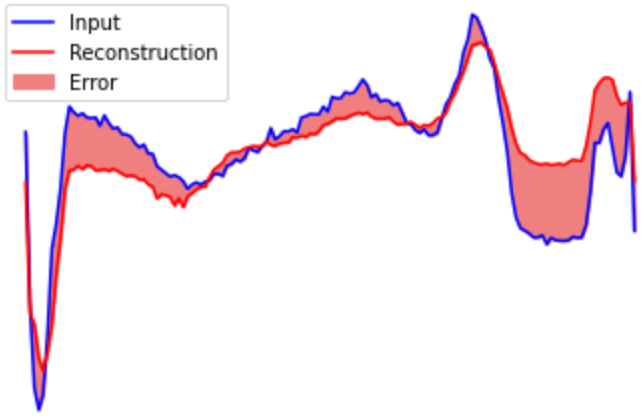

Fig 3: Normal ECG, Autoencoder Reconstruction, and Reconstruction Error

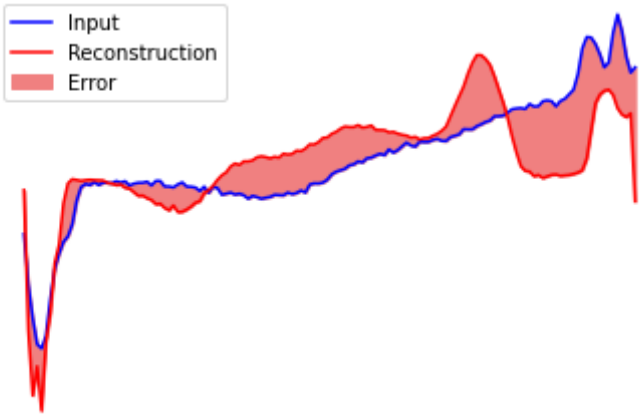

Fig. 4: Abnormal ECG, Autoencoder Reconstruction, and Reconstruction Error

Finally, a loss threshold is determined with the sum of the MEAN(training loss) and STANDARD DEVIATION (training loss). This threshold is used to predict labels: normal and abnormal for the full data set. The performance for the base model is as follows:

Table 1: Baseline Model Performance

Accuracy	0.944
Precision	0.992
Recall	0.907

F1 Score	0.948

Optimization of the solution

Efforts to optimize the solution presented in the previous section included the following:

1. Design & Training: Batch size decreased from 512 to 42
2. Design & Training: Epochs increased from 20 to 30
3. Evaluation & Optimization: Optimizer changed from Adam to Adamax
4. Deployment & Model Inference: Changed threshold value from one to two times the standard deviation of training loss

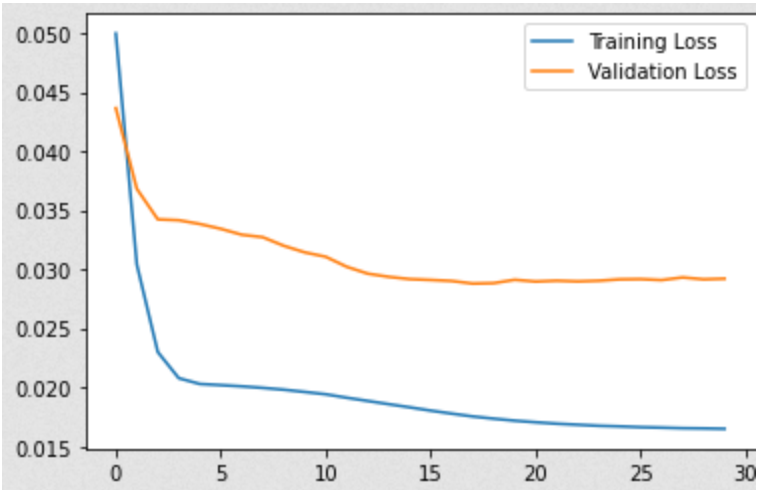

Fig. 5: Optimized Training Loss and Validation Loss

A lower batch size may lead to noisier data, but it greatly improved the training time per epoch, allowing for more epochs to be trained with a slight increase in training time. AdaMax optimizer is an extension to the Adam version of gradient descent that generalizes the approach to the infinite norm, designed to accelerate the optimization process. Increasing the loss threshold value had a nominal negative impact on precision, but all the optimizations employed improved Accuracy, Recall, and F1 score - also increasing validation accuracy by 3.2%.

Table 2: Baseline and optimization comparison

Metric	**Baseline Model**	**Optimized Model**

Training time	4 s	6 s
Validation Loss	0.033	0.026
Loss threshold	0.033	0.027
Accuracy	0.944	0.976
Precision	0.992	0.990
Recall	0.907	0.966
F1	0.948	0.978

Resources

1. Intro to Autoencoders, Tensorflow
    a. https://www.tensorflow.org/tutorials/generative/autoencoder
2. ECG5000 Dataset, Time Series Classification
    a. http://www.timeseriesclassification.com/description.php?Dataset=ECG5000