In this tutorial I will discuss on how to use [keras package](https://keras.rstudio.com/) with tensor flow as back end to build an anomaly detection model using auto encoders. A[uto encoders is a unsupervised learning technique](https://en.wikipedia.org/wiki/Autoencoder) where the initial data is encoded to lower dimensional and then decoded (reconstructed) back. Based on our initial data and reconstructed data we will calculate the score.

**About Dataset**

The data is contains 66 features extracted from a [vibration signal](https://en.wikipedia.org/wiki/Vibration) from x, y & z axis. For the experiment, a 3 axis vibration sensor was hooked up to a table press drill. There are total for 4 failure modes within the data set. This data also as numeric and categorical labels.

**Load the libraries**

# load the libraries

library(keras)

library(dplyr)

**Load data set to R**

The data is loaded and the labels are removed. The data set is split into train and test. Train includes calibration data and test includes remaining data set. The data is converted to matrix as required by keras package.

# load the data set

data = read.csv("features.csv", header = T) %>%

select(-c(yLabel, Y)) %>%

as.data.frame()

# convert all to numeric

data = sapply(data, as.numeric)

# split the data in to train and test

train = data[1:50,] %>% as.matrix()

test = data[51:357,] %>% as.matrix()

**Set parameters for auto encoder model**

We are creating a set of parameters below. This is optional. But, it makes it easy for hyper parameter tuning.

dropOut = 0.2

atvn = "sigmoid"

batch = 10

**Auto encoder model**

The auto encoder model used here is a symmetric model.  
Input layer: Includes the train shape of the data. ie, total of 66 features.

# create auto encoder architecture

input\_layer =

layer\_input(shape = c(66))

encoder =

input\_layer %>%

layer\_dense(units = 512, activation = atvn) %>%

layer\_batch\_normalization() %>%

layer\_dropout(rate = dropOut) %>%

layer\_dense(units = 128, activation = atvn) %>%

layer\_dropout(rate = dropOut) %>%

layer\_dense(units = 64, activation = atvn) %>%

layer\_dense(units = 32)

decoder =

encoder %>%

layer\_dense(units = 64, activation = atvn) %>%

layer\_dropout(rate = dropOut) %>%

layer\_dense(units = 128, activation = atvn) %>%

layer\_dropout(rate = dropOut) %>%

layer\_dense(units = 512, activation = atvn) %>%

layer\_dense(units = 66) #

**Training**

Next, we combine our input layer and decoder to form a auto encoder model. Next, we compile the model with different optimizer and loss function. Finally we can fit the model and plot the results.

# combine encoder and decoder layers

autoencoder\_model = keras\_model(inputs = input\_layer,

outputs = decoder)

# compile the model

autoencoder\_model %>% compile(

loss='mean\_squared\_error',

optimizer='adam'

)

# look at the summary of the model

summary(autoencoder\_model)

# fit the model

history =

autoencoder\_model %>%

keras::fit(train,

train,

epochs=100,

shuffle=TRUE,

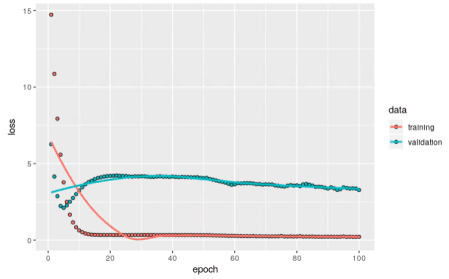
batch = batch,

validation\_data= list(test, test)

)

# view the history

plot(history)



*Plot history for training the model*

**Reconstruction error and anomaly limits**

We have a function calculate the [reconstruction error](https://www.jeremyjordan.me/autoencoders/) and based on train data set, we will use 85% quantile to set the anomaly limit. We finally combine the data set to plot the results. Below, we see all green is healthy data points and red is abnormal condition.

# function to calculate reconstruction error

reconstMSE = function(i){

reconstructed\_points = autoencoder\_model %>%

predict(x = data[i,] %>%

matrix(nrow = 1, ncol = 66)

)

return(mean((data[i,] - reconstructed\_points)^2))

}

# inital data is train

data = train

# calculate reconstruction error

trainRecon = data.frame(data = train,

score = do.call(rbind,

lapply(1:50,

FUN = reconstMSE)

)

)

# calculate anomaly limit

anomalyLimit = quantile(trainRecon$score, p = 0.85)

# next, test data

data = test

# calculate test reconstruction error

testRecon = data.frame(data = test,

score = do.call(rbind,

lapply(1:nrow(data),

FUN = reconstMSE)

)

)

# combine train and test errors

Recondata = rbind(trainRecon, testRecon)

# plot the results

plot(Recondata$score,

col = ifelse(Recondata$score>anomalyLimit, "red", "green"),

pch = 19,

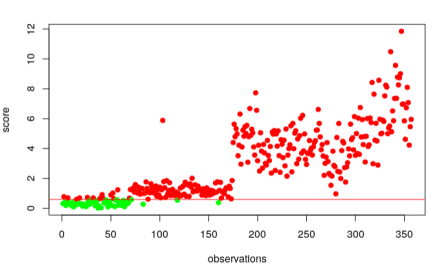
xlab = "observations",

ylab = "score")

abline(h = anomalyLimit,

col = "red",

lwd = 1)



*Anomaly detection results for the entire data set*

From the above result, we observer that we have few false positives. But, we could tune the parameters and retrain them to achieve higher accuracy.