Autoencoders is an unsupervised version of neural network that is used for data encoding. This technique is mainly used to learn the representation of data that can be used for dimensionality reduction by training network to ignore noise. Autoencoders play an important role in unsupervised learning and deep architectures mainly for transfer learning (Pierre. B, 2012). When autoencoders are decoded, they are simple linear circuits that transforms inputs to outputs with least distortion. Autoencoders were first introduced in 1980’s to address the issue of back propagation without training and rather use input as a teacher (Rumelhart et al., 1986). Since then, autoencoders have taken a phase change to the form on Restricted Boltzman Machine. Today, autoencoders are used in various applications such as predicting sentiment distributions in Natural Language Processing (NLP) (Socher et al., 2011a) (Socher et al., 2011b), feature extraction (Masci et al., 2011), anomaly detection (Sakurada et al., 2014), facial recognition (Gao et al., 2015), clustering (Dilokthanakul et al., 2016), image classification (Geng et al., 2015) and many other application.

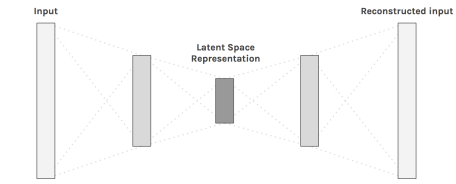


Image: Simple auto encoder representation

In today’s tutorial, I will go over on how to use auto encoders for anomaly detection in predictive maintenance.

**Load Libraries**

You will need only two libraries for this analysis.

options(warn=-1)

# load libraries

library(dplyr)

library(h2o)

**Load data**

Here we are using data from a bench press. There are total of four different states in this machine and they are split into four different csv files. We need to load the data first. In the data time represents the time between samples, ax is the acceleration on x axis, ay is the acceleration on y axis, az is the acceleration on z axis and at is the G’s. The data was collected at sample rate of 100hz.

Four different states of the machine were collected

1. Nothing attached to drill press

2. Wooden base attached to drill press

3. Imbalance created by adding weight to one end of wooden base

4. Imbalance created by adding weight to two ends of wooden base.

setwd("/home/")

#read csv files

file1 = read.csv("dry run.csv", sep=",", header =T)

file2 = read.csv("base.csv", sep=",", header =T)

file3 = read.csv("imbalance 1.csv", sep=",", header =T)

file4 = read.csv("imbalance 2.csv", sep=",", header =T)

head(file1)

| **time** | **ax** | **ay** | **az** | **aT** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0.002 | -0.3246 | 0.2748 | 0.1502 | 0.451 |
| 0.009 | 0.6020 | -0.1900 | -0.3227 | 0.709 |
| 0.019 | 0.9787 | 0.3258 | 0.0124 | 1.032 |
| 0.027 | 0.6141 | -0.4179 | 0.0471 | 0.744 |
| 0.038 | -0.3218 | -0.6389 | -0.4259 | 0.833 |
| 0.047 | -0.3607 | 0.1332 | -0.1291 | 0.406 |

We can look at the summary of each file using summary function in R. Below, we can observe that 66 seconds long data is available. We also have min, max and mean for each of the variables.

# summary of each file

summary(file2)

time ax ay az

Min. : 0.004 Min. :-1.402700 Min. :-1.693300 Min. :-3.18930

1st Qu.: 27.005 1st Qu.:-0.311100 1st Qu.:-0.429600 1st Qu.:-0.57337

Median : 54.142 Median : 0.015100 Median :-0.010700 Median :-0.11835

Mean : 54.086 Mean : 0.005385 Mean :-0.002534 Mean :-0.09105

3rd Qu.: 81.146 3rd Qu.: 0.314800 3rd Qu.: 0.419475 3rd Qu.: 0.34815

Max. :108.127 Max. : 1.771900 Max. : 1.515600 Max. : 5.04610

aT

Min. :0.0360

1st Qu.:0.6270

Median :0.8670

Mean :0.9261

3rd Qu.:1.1550

Max. :5.2950

**Data Aggregation and feature extraction**

Here, the data is aggregated by 1 minute and features are extracted. Features are extracted to reduce the size of the data and only storing the representation of the data.

file1$group = as.factor(round(file1$time))

file2$group = as.factor(round(file2$time))

file3$group = as.factor(round(file3$time))

file4$group = as.factor(round(file4$time))

#(file1,20)

#list of all files

files = list(file1, file2, file3, file4)

#loop through all files and combine

features = NULL

for (i in 1:4){

res = files[[i]] %>%

group\_by(group) %>%

summarize(ax\_mean = mean(ax),

ax\_sd = sd(ax),

ax\_min = min(ax),

ax\_max = max(ax),

ax\_median = median(ax),

ay\_mean = mean(ay),

ay\_sd = sd(ay),

ay\_min = min(ay),

ay\_may = max(ay),

ay\_median = median(ay),

az\_mean = mean(az),

az\_sd = sd(az),

az\_min = min(az),

az\_maz = max(az),

az\_median = median(az),

aT\_mean = mean(aT),

aT\_sd = sd(aT),

aT\_min = min(aT),

aT\_maT = max(aT),

aT\_median = median(aT)

)

features = rbind(features, res)

}

#view all features

head(features)

**Create Train and Test Set**

To build an anomaly detection model, a train and test set is required. Here, the normal condition of the data is used for training and remaining is used for testing.

# create train and test set

train = features[1:67,2:ncol(features)]

test = features[68:nrow(features),2:ncol(features)]

**Auto Encoders**

**Auto Encoders using H2O package**

Use the [h2o.init()](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/starting-h2o.html)method to initialize H2O. This method accepts the following options. Note: that in most cases, simply using h2o.init() is all that a user is required to do.

# initialize h2o cluser

h2o.init()

The R object to be converted to an H2O object should be named so that it can be used in subsequent analysis. Also note that the R object is converted to a parsed H2O data object, and will be treated as a data frame by H2O in subsequent analysis.

# convert train and test to h2o object

train\_h2o = as.h2o(train)

test\_h2o = as.h2o(test)

The [h2o.deeplearning](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html) function fits H2O’s Deep Learning models from within R. While H2O Deep Learning has many parameters, it was designed to be just as easy to use as the other supervised training methods in H2O. Early stopping, automatic data standardization and handling of categorical variables and missing values and adaptive learning rates (per weight) reduce the amount of parameters the user has to specify. Often, it’s just the number and sizes of hidden layers, the number of epochs and the activation function and maybe some regularization techniques.

# build auto encoder model with 3 layers

model\_unsup = h2o.deeplearning(x = 2:ncol(features)

, training\_frame = train\_h2o

, model\_id = "Test01"

, autoencoder = TRUE

, reproducible = TRUE

, ignore\_const\_cols = FALSE

, seed = 42

, hidden = c(50,10,50,100,100)

, epochs = 100

, activation ="Tanh")

# view the model

model\_unsup

Model Details:

==============

H2OAutoEncoderModel: deeplearning

Model ID: Test01

Status of Neuron Layers: auto-encoder, gaussian distribution, Quadratic loss, 19,179 weights/biases, 236.0 KB, 2,546 training samples, mini-batch size 1

layer units type dropout l1 l2 mean\_rate rate\_rms momentum

1 1 19 Input 0.00 % NA NA NA NA NA

2 2 50 Tanh 0.00 % 0.000000 0.000000 0.029104 0.007101 0.000000

3 3 10 Tanh 0.00 % 0.000000 0.000000 0.021010 0.006320 0.000000

4 4 50 Tanh 0.00 % 0.000000 0.000000 0.024570 0.006848 0.000000

5 5 100 Tanh 0.00 % 0.000000 0.000000 0.052482 0.018357 0.000000

6 6 100 Tanh 0.00 % 0.000000 0.000000 0.052677 0.021417 0.000000

7 7 19 Tanh NA 0.000000 0.000000 0.025557 0.009494 0.000000

mean\_weight weight\_rms mean\_bias bias\_rms

1 NA NA NA NA

2 0.000069 0.180678 0.001542 0.017311

3 0.000008 0.187546 -0.000435 0.011542

4 0.011644 0.184633 0.000371 0.006443

5 0.000063 0.113350 -0.000964 0.008983

6 0.000581 0.100150 0.001003 0.013848

7 -0.001349 0.121616 0.006549 0.012720

H2OAutoEncoderMetrics: deeplearning

\*\* Reported on training data. \*\*

Training Set Metrics:

=====================

MSE: (Extract with `h2o.mse`) 0.005829827

RMSE: (Extract with `h2o.rmse`) 0.0763533

Detect anomalies in an H2O data set using an H2O deep learning model with auto-encoding trained previously.

# now we need to calculate MSE or anomaly score

anmlt = h2o.anomaly(model\_unsup

, train\_h2o

, per\_feature = FALSE) %>% as.data.frame()

# create a label for healthy data

anmlt$y = 0

# view top data

head(anmlt)

| **Reconstruction.MSE** | **y** |
| --- | --- |
|  |  |
| 0.001953387 | 0 |
| 0.004875430 | 0 |
| 0.002195593 | 0 |
| 0.006722837 | 0 |
| 0.001670331 | 0 |
| 0.005859846 | 0 |

Calculate the threshold value for train[anomaly scores](https://www.rdocumentation.org/packages/h2o/versions/3.26.0.2/topics/h2o.anomaly?tap_a=5644-dce66f&tap_s=10907-287229). Various methods can be used such as calculating the quantiles, max, median, min etc. It all depends on the use case. Here we will use quantile with probability of 99.9%.

# calculate thresholds from train data

threshold = quantile(anmlt$Reconstruction.MSE, probs = 0.999)

Now, we have anomaly score for train and its thresholds, we can predict the new anomaly scores for test data and plot it to see how it differs from train data.

# calculate anomaly scores for test data

test\_anmlt = h2o.anomaly(model\_unsup

, test\_h2o

, per\_feature = FALSE) %>% as.data.frame()

# create a label for healthy data

test\_anmlt$y = 1

# combine the train and test anomaly scores for visulaizatio

results = data.frame(rbind(anmlt,test\_anmlt), threshold)

head(results)

| **Reconstruction.MSE** | **y** | **threshold** |
| --- | --- | --- |
|  |  |  |
| 0.001953387 | 0 | 0.01705935 |
| 0.004875430 | 0 | 0.01705935 |
| 0.002195593 | 0 | 0.01705935 |
| 0.006722837 | 0 | 0.01705935 |
| 0.001670331 | 0 | 0.01705935 |
| 0.005859846 | 0 | 0.01705935 |

The results are plotted below. The x axis is the observations and y axis is the anomaly score. The green points are the trained data and red are test data. We can note that all the data that was trained except one lied below the anomaly limit. Its also interesting to note the increasing trend pattern for the anomaly scores for other state of the machine.

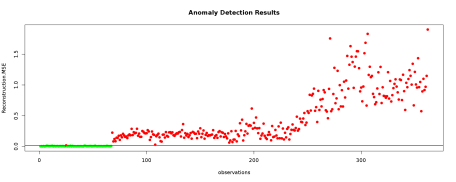
# Adjust plot sizes

options(repr.plot.width = 15, repr.plot.height = 6)

plot(results$Reconstruction.MSE, type = 'n', xlab='observations', ylab='Reconstruction.MSE', main = "Anomaly Detection Results")

points(results$Reconstruction.MSE, pch=19, col=ifelse(results$Reconstruction.MSE < threshold, "green", "red"))

abline(h=threshold, col='red', lwd=2)



**Conclusion**

Auto encoder is a very powerful tool and very fun to play with. They have been used in image analysis, image reconstruction and [image colorization](https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694). In this tutorial you have seen how to perform anomaly detection on a simple signal data and few lines of code. The possibilities of using this are many. Let me know what you think about auto encoders in the comments below.

**Session info**

Below is the session info for the the packages and their versions used in this analysis.

sessionInfo()

R version 3.3.3 (2017-03-06)

Platform: x86\_64-pc-linux-gnu (64-bit)

Running under: Debian GNU/Linux 9 (stretch)

locale:

[1] LC\_CTYPE=C.UTF-8 LC\_NUMERIC=C LC\_TIME=C.UTF-8

[4] LC\_COLLATE=C.UTF-8 LC\_MONETARY=C.UTF-8 LC\_MESSAGES=C.UTF-8

[7] LC\_PAPER=C.UTF-8 LC\_NAME=C LC\_ADDRESS=C

[10] LC\_TELEPHONE=C LC\_MEASUREMENT=C.UTF-8 LC\_IDENTIFICATION=C

attached base packages:

[1] stats graphics grDevices utils datasets methods base

other attached packages:

[1] h2o\_3.26.0.2 dplyr\_0.8.3

loaded via a namespace (and not attached):

[1] Rcpp\_1.0.2 magrittr\_1.5 tidyselect\_0.2.5

[4] uuid\_0.1-2 R6\_2.4.0 rlang\_0.4.0

[7] tools\_3.3.3 htmltools\_0.3.6 assertthat\_0.2.1

[10] digest\_0.6.20 tibble\_2.1.3 crayon\_1.3.4

[13] IRdisplay\_0.7.0 purrr\_0.3.2 repr\_1.0.1

[16] base64enc\_0.1-3 vctrs\_0.2.0 bitops\_1.0-6

[19] RCurl\_1.95-4.12 IRkernel\_1.0.2.9000 zeallot\_0.1.0

[22] glue\_1.3.1 evaluate\_0.14 pbdZMQ\_0.3-3

[25] pillar\_1.4.2 backports\_1.1.4 jsonlite\_1.6

[28] pkgconfig\_2.0.2