Accounting Anomaly Detector

Executive Summary

The company is authorised by the Financial Conduct Authority, which requires it to have regular processes to identify potential fraud. There are millions of records in the accounting system, too many to manually review. An autoencoder is developed which very accurately sifts unusual transactions from those which are commonplace. It is sufficiently accurate to flag 100 intentionally unusual transactions from 250,000 test transactions, with almost no false positives.

The autoencoder also indicates which component of a record causes it to be flagged as abnormal, ie it provides some degree of explanation.

This work is inspired by a paper by PwC (https://arxiv.org/pdf/1709.05254.pdf) and adapted for the company's data.

Introduction

The company has an accounting system with approx 250,000 debits and credits made per year. Detecting fraud or error within this mass of data is not straightforward. Traditional solutions involve querying the data for 'red flags', transactinos at odd times of day for example. A more recent approach is the autoencoder, a deep learning tool which attempts to compress the information within a record, then reconstitute it into a copy of the original record. Since the vast majority of record sin a system will be 'business as usual', then the autoencoder can learn to re-constitute them well. It will be less well trained at unusual records, i.e. fraud or error, so these records are less well reconstituted. The delta between the original copy and its recreation is considered a measure of its un-usualness.

The company's accounting system has been in place for 10years, meaning there are now over 2,000,000 records in the database. This should easily be enough for a robust deep learning model to be built.

```r
library(dplyr)
library(tidyr)
library(RODBC)
library(reticulate)
library(knitr)

# Lots of packages, ensure dplyr gets priority
select    <- dplyr::select
filter    <- dplyr::filter
summarise <- dplyr::summarise

# Extract the required cash flow data.
# The SQL used in the stored procedure is in an appendix at the end of the notebook
srcData <- sqlQuery(connection, "EXEC [sp_GetAllNominalTransactions]", stringsAsFactors=FALSE)

# Close connection
odbcClose(connection)

#Let's see how big this data is
paste0(format(object.size(srcData), units = "auto"))
```

Lets take a look at the data, which is randomly sorted already…

```r
str(srcData)
```

```
## 'data.frame':    2067405 obs. of  23 variables:
##  $ ACCOUNT_NO_Digit1: int  2 1 2 1 1 4 2 4 4 3 ...
##  $ ACCOUNT_NO_Digit2: int  4 7 4 6 7 7 1 7 7 7 ...
##  $ ACCOUNT_NO_Digit3: int  7 2 7 0 2 2 5 2 2 2 ...
##  $ ACCOUNT_NO_Digit4: int  0 0 0 0 0 0 0 0 3 2 ...
##  $ ACCOUNT_NO_Digit5: int  1 1 1 1 1 2 1 2 8 1 ...
##  $ ACCOUNT_NO_Digit6: int  0 0 0 0 0 2 0 2 2 2 ...
##  $ ACCOUNT_NO_Digit7: int  0 2 0 1 2 3 0 2 1 3 ...
##  $ TRANS_DATE       : Date, format: "2013-05-25" "2016-10-15" ...
##  $ TRANS_DATE_Mth   : int  5 10 4 1 6 1 9 1 2 7 ...
##  $ TRANS_DATE_DoM   : int  25 15 13 25 20 5 23 20 21 31 ...
##  $ TRANS_DATE_DoW   : int  7 7 6 3 3 5 2 5 4 5 ...
##  $ SYSTEM_DATE      : Date, format: "2013-05-25" "2016-10-17" ...
##  $ SYSTEM_DATE_Mth  : int  5 10 4 1 6 1 10 1 2 8 ...
##  $ SYSTEM_DATE_DoM  : int  25 17 13 25 20 5 14 21 21 4 ...
##  $ SYSTEM_DATE_DoW  : int  7 2 6 3 3 5 2 6 4 2 ...
##  $ PostingDelayDays : int  0 2 0 0 0 0 21 1 0 4 ...
##  $ DOCUMENT_REF_L1  : chr  "4" "2" "1" "6" ...
##  $ BC_TRANS_VALUE   : num  -0.83 -337.49 -140 126.58 20.14 ...
##  $ TRANS_TYPE       : chr  "IN" "IN" "IN" "PI" ...
##  $ OPERATOR         : chr  "ML" "KD" "KD" "MM" ...
##  $ CUST_REF         : int  1004 2904 8004 -1 -1 1003 -1 -1 938 4679 ...
##  $ SUPP_REF         : int  -1 -1 -1 -1 -1 -1 1259 -1 -1 -1 ...
##  $ DOCUMENT_REF     : chr  "462581" "265245" "155761" "61908" ...
```

So each row is a posting to a nominal, either a debit (+ve) or a credit (-ve). The system groups a transaction into the component debits and credits which normally balance to zero. So there may be multiple rows (postings) per a single transaction. The key by which we group the postings into a transaction is the 'DOCUMENT_REF'. This data has only the first digit of the DOCUMENT_REF, which holds information, for example 'P' is the first digit of a purchase, sales are prefixed with the number associated with a depot (1 to 5). Subsequent digits are simply a sequential number, so its no loss to have them omitted from the data.

The nominal 'account' number is strictly a categorical field, although it appears numerical. The first digit is the type of the account, 1= Asset, 2= Liability , 3 = Sale etc. The last digit denotes the depot of the transaction; 0 = Not depot specific, 1= Telford, 2=Hereford, etc. These are separated into their own columns.

There are thousands of nominal accounts, which would lead to a one-hot vector thousands wide, SInce each digit has some meaning, we'll create a one-hot vecot for each of the 7 digits in a nominal account number. Meaning only 10x7=70 columns for the one-hot of an entire nominal account number.

This data has been randomly sorted so the postings of each transaction are not listed together.

Data Cleaning & Prep

Much of the data is comprised of categories, these need to be facorised appropoiatey and a record kept of the factor levels.

```r
# Column specification will be used frequently during this analysis, so best presented as a table
Col_Spec <- cbind.data.frame(Name      = colnames(srcData),
                             Type_init = as.vector(sapply(srcData, class)),
                             Type_final= as.character(NA),
                             Min       = as.numeric(NA),
                             Max       = as.numeric(NA),
```

```r
                             MinMaxed  = FALSE,
                             Logged    = FALSE,
                             stringsAsFactors = FALSE)

# get a copy of the source data, we will adjust only this copy, not the original
srcData_adj <- srcData

# This is the kind of data where it helps to set NA to zero
srcData_adj[is.na(srcData_adj)] <- 0

#Create factors where data is text
srcData_adj$ACCOUNT_NO_Digit1<- as.factor(srcData_adj$ACCOUNT_NO_Digit1)
srcData_adj$ACCOUNT_NO_Digit2<- as.factor(srcData_adj$ACCOUNT_NO_Digit2)
srcData_adj$ACCOUNT_NO_Digit3<- as.factor(srcData_adj$ACCOUNT_NO_Digit3)
srcData_adj$ACCOUNT_NO_Digit4<- as.factor(srcData_adj$ACCOUNT_NO_Digit4)
srcData_adj$ACCOUNT_NO_Digit5<- as.factor(srcData_adj$ACCOUNT_NO_Digit5)
srcData_adj$ACCOUNT_NO_Digit6<- as.factor(srcData_adj$ACCOUNT_NO_Digit6)
srcData_adj$ACCOUNT_NO_Digit7<- as.factor(srcData_adj$ACCOUNT_NO_Digit7)
srcData_adj$TRANS_TYPE       <- as.factor(srcData_adj$TRANS_TYPE)
srcData_adj$OPERATOR         <- as.factor(srcData_adj$OPERATOR)
srcData_adj$DOCUMENT_REF_L1  <- as.factor(srcData_adj$DOCUMENT_REF_L1)
srcData_adj$SUPP_REF         <- as.factor(srcData_adj$SUPP_REF)
srcData_adj$CUST_REF         <- as.factor(srcData_adj$CUST_REF)

## get the levels for future reference
ACCOUNT_NO_Digit1_Levels    <- levels(srcData_adj$ACCOUNT_NO_Digit1)
ACCOUNT_NO_Digit2_Levels    <- levels(srcData_adj$ACCOUNT_NO_Digit2)
ACCOUNT_NO_Digit3_Levels    <- levels(srcData_adj$ACCOUNT_NO_Digit3)
ACCOUNT_NO_Digit4_Levels    <- levels(srcData_adj$ACCOUNT_NO_Digit4)
ACCOUNT_NO_Digit5_Levels    <- levels(srcData_adj$ACCOUNT_NO_Digit5)
ACCOUNT_NO_Digit6_Levels    <- levels(srcData_adj$ACCOUNT_NO_Digit6)
ACCOUNT_NO_Digit7_Levels    <- levels(srcData_adj$ACCOUNT_NO_Digit7)
TRANS_TYPE_Levels           <- levels(srcData_adj$TRANS_TYPE)
OPERATOR_Levels             <- levels(srcData_adj$OPERATOR)
DOCUMENT_REF_L1_Levels      <- levels(srcData_adj$DOCUMENT_REF_L1)
SUPP_REF_Levels             <- levels(srcData_adj$SUPP_REF)
CUST_REF_Levels             <- levels(srcData_adj$CUST_REF)

# identify the columns which are factors
ColIsFactor <- seq(1:ncol(srcData_adj))[sapply(srcData_adj, function(x) is.factor(x))]

# add this knowledge to our column specification table
Col_Spec[ColIsFactor,]$Type_final  <- "factor"
Col_Spec[-ColIsFactor,]$Type_final <- "numerical"

# collate levels into one object, for future use
factorlevels <- list(as.numeric(ACCOUNT_NO_Digit1_Levels), as.numeric(ACCOUNT_NO_Digit2_Levels),
                     as.numeric(ACCOUNT_NO_Digit3_Levels), as.numeric(ACCOUNT_NO_Digit4_Levels),
                     as.numeric(ACCOUNT_NO_Digit5_Levels), as.numeric(ACCOUNT_NO_Digit6_Levels),
                     as.numeric(ACCOUNT_NO_Digit7_Levels), DOCUMENT_REF_L1_Levels,
                     TRANS_TYPE_Levels, OPERATOR_Levels, SUPP_REF_Levels, CUST_REF_Levels)

# apply names to the object storing our levels
```

```r
names(factorlevels) <- colnames(srcData_adj)[ColIsFactor]

## convert each categorical field to the numeric value of the levels, as required by the deep learning
srcData_adj$ACCOUNT_NO_Digit1<- as.numeric(srcData_adj$ACCOUNT_NO_Digit1)
srcData_adj$ACCOUNT_NO_Digit2<- as.numeric(srcData_adj$ACCOUNT_NO_Digit2)
srcData_adj$ACCOUNT_NO_Digit3<- as.numeric(srcData_adj$ACCOUNT_NO_Digit3)
srcData_adj$ACCOUNT_NO_Digit4<- as.numeric(srcData_adj$ACCOUNT_NO_Digit4)
srcData_adj$ACCOUNT_NO_Digit5<- as.numeric(srcData_adj$ACCOUNT_NO_Digit5)
srcData_adj$ACCOUNT_NO_Digit6<- as.numeric(srcData_adj$ACCOUNT_NO_Digit6)
srcData_adj$ACCOUNT_NO_Digit7<- as.numeric(srcData_adj$ACCOUNT_NO_Digit7)
srcData_adj$TRANS_TYPE       <- as.numeric(srcData_adj$TRANS_TYPE)
srcData_adj$OPERATOR         <- as.numeric(srcData_adj$OPERATOR)
srcData_adj$DOCUMENT_REF_L1  <- as.numeric(srcData_adj$DOCUMENT_REF_L1)
# EXCEPT CUST_REF and SUPP_REF, dealt with in next codechunk

#Set dates to numeric
srcData_adj$TRANS_DATE       <- as.numeric(srcData_adj$TRANS_DATE)
srcData_adj$SYSTEM_DATE      <- as.numeric(srcData_adj$SYSTEM_DATE)

# NOTE, do nothing with DOCUMENT_REF. It will NOT be fed to the model, is unique to each transaction
# It will be used in testing, to link transactions back to documents in the accts system
# Thus enabling us to investigate candidate records for error and even fraud
# For now, we will simply record its function in our Col_Spec table.
# Note Type_final = "exclude", ie it will be excluded from the model.
#Remove the automatically created entry
Col_Spec <- Col_Spec %>% filter(Name != "DOCUMENT_REF")
#Replace with this entry
Col_Spec[nrow(Col_Spec)+1,] <- list("DOCUMENT_REF", "character", "exclude",
                                    as.numeric(NA), as.numeric(NA), FALSE, FALSE)
```

CUST_REF and SUPP_REF

These two fields are factors with thousands of levels, this would lead to enormous onehot vectors later in the analysis. So, we will use only the top 100 factors and lump the rest into 'other'. The tidyverse package, 'forcats', has tools for this job. The top 100 will capture a large proportion of the data, let's see the count per factor, after the 'lumping' together…

```r
library(forcats)

#100 levels, 1 'other' of lumped together rarest factors, and 1 N/A (-1)
srcData_adj$CUST_REF <- fct_lump(srcData_adj$CUST_REF, n = 100) #101 levels = 100 factors + 'other'
srcData_adj$SUPP_REF <- fct_lump(srcData_adj$CUST_REF, n = 100) #101 levels = 100 factors + 'other'

#record the new levels
CUST_REF_Levels      <- levels(srcData_adj$CUST_REF)
SUPP_REF_Levels      <- levels(srcData_adj$SUPP_REF)

#Let's see the countÃ¼
fct_count(srcData_adj$CUST_REF) %>% arrange(desc(n))

## convert each categorical field to the numeric value of the levels, as required by the deep learning
srcData_adj$SUPP_REF <- as.numeric(srcData_adj$SUPP_REF)
```

```
srcData_adj$CUST_REF <- as.numeric(srcData_adj$CUST_REF)
```

New Field for Debits vs Credits

Currently the BC_TRANS_VALUE is +ve for a debit and -ve for a credit. It will later prove useful to separate the sign of the transaction from the value. Before we do, we should see the distribution of debits vs credits:

```
debit_count  <- sum(srcData_adj$BC_TRANS_VALUE >= 0)
credit_count <- sum(srcData_adj$BC_TRANS_VALUE <  0)
```

```
print(paste0("For each debit there are ",round(credit_count/debit_count,2), " credits"))
```

```
## [1] "For each debit there are 1.19 credits"
```

A slight skew to credits. Now we adjust the data for BC_TRANS_VALUE into two columns, a BC_TRANS_VALUE which is always +ve and a separate column for the debit or credit sign. Debit (+ve) = 1 , Credit (-ve) = 0

```
srcData_adj <- srcData_adj %>% mutate(isDebit = ifelse(BC_TRANS_VALUE >= 0,1,0))
srcData_adj <- srcData_adj %>% mutate(BC_TRANS_VALUE = abs(BC_TRANS_VALUE))

#We treat the isDebit field as binary, we won't need to scale it later.
Col_Spec[nrow(Col_Spec)+1,] <- list("isDebit", "binary", "binary",
                                    as.numeric(NA),as.numeric(NA), FALSE, FALSE)
```

Create Dummy Data

We currently don't know whether our data contains any true frauds, we're not aware of any! So how can we test whether the model is properly identifying unusual records?

We can create 1000 records which are simply randomly created. Since they will make no no sense we'd hope the autoencoder can find them.

```
library(lubridate)

dummyqty <- 1000

get_random <- function(range, isFactor, isTransValue = FALSE, isPostingDelay = FALSE,
                       dummy_qty = dummyqty, seed = seed){

  #convert levels to values
  if(isFactor){
    range <- seq_along(range)
  }
  #get max and min
  max_in_range <- max(range)
  min_in_range <- min(range)

  #if we're dealing with the transaction value, try not to take more than Â£50k, bit too obvious!
  if(isTransValue){
    max_in_range <-  50000
    min_in_range <-  0
  }
```

```
  # if we're dealing with the posting delay then it should be within -2 to +30 days
  if(isPostingDelay){
    max_in_range <-  30
    min_in_range <-  -2
  }

  #use uniform probability
  set.seed(seed)
  randoms <- runif(dummy_qty, min_in_range, max_in_range)

  #round all values to integer, EXCEPT TransValue which is rounded to 2 places
  if(isTransValue){
    randoms <- round(randoms,2)
  }else{
    randoms <- round(randoms)
  }

  return(randoms)
}

dummy_rows <- cbind.data.frame(
                 ACCOUNT_NO_Digit1 = get_random(range=ACCOUNT_NO_Digit1_Levels,
                                                isFactor=TRUE, seed=20181208),
                 ACCOUNT_NO_Digit2 = get_random(ACCOUNT_NO_Digit2_Levels,
                                                isFactor=TRUE, seed=20181209),
                 ACCOUNT_NO_Digit3 = get_random(ACCOUNT_NO_Digit3_Levels,
                                                isFactor=TRUE, seed=20181210),
                 ACCOUNT_NO_Digit4 = get_random(ACCOUNT_NO_Digit4_Levels,
                                                isFactor=TRUE, seed=20181211),
                 ACCOUNT_NO_Digit5 = get_random(ACCOUNT_NO_Digit5_Levels,
                                                isFactor=TRUE, seed=20181212),
                 ACCOUNT_NO_Digit6 = get_random(ACCOUNT_NO_Digit6_Levels,
                                                isFactor=TRUE, seed=20181213),
                 ACCOUNT_NO_Digit7 = get_random(ACCOUNT_NO_Digit7_Levels,
                                                isFactor=TRUE, seed=20181214),
                 TRANS_DATE        = get_random(srcData_adj$TRANS_DATE,
                                                isFactor=FALSE,seed=20181215),

                 # These TRANS_DATE fields are dictated by the TRANS_DATE, so calculated later
                 TRANS_DATE_Mth    = 0,
                 TRANS_DATE_DoM    = 0,
                 TRANS_DATE_DoW    = 0,

                 # These SYSTEM_DATE fields are dictated by the PostingDelayDays, so calculated later
                 SYSTEM_DATE       = 0,
                 SYSTEM_DATE_Mth   = 0,
                 SYSTEM_DATE_DoM   = 0,
                 SYSTEM_DATE_DoW   = 0,

                 PostingDelayDays  = get_random(srcData_adj$PostingDelayDays,isFactor=FALSE,
                                                isPostingDelay=TRUE, seed=20181216),
                 DOCUMENT_REF_L1   = get_random(DOCUMENT_REF_L1_Levels,      isFactor=TRUE,
                                                seed=20181217),
```

```
                 BC_TRANS_VALUE    = get_random(srcData_adj$BC_TRANS_VALUE,  isFactor=FALSE,
                                                isTransValue=TRUE,    seed=20181218),
                 TRANS_TYPE        = get_random(TRANS_TYPE_Levels,           isFactor=TRUE,
                                                 seed=20181219),
                 OPERATOR          = get_random(OPERATOR_Levels,             isFactor=TRUE,
                                                 seed=20181220),
                 CUST_REF          = get_random(CUST_REF_Levels,             isFactor=TRUE,
                                                 seed=20181221),
                 SUPP_REF          = get_random(SUPP_REF_Levels,             isFactor=TRUE,
                                                 seed=20181222),

                 # NOTE: DOCUMENT_REF will not be fed to the model, so leave as NA
                 DOCUMENT_REF       = NA,

                 # Flag that the record is randomly created for testing, not a real record.
                 isRandom          = TRUE,

                 # Flag for debit or Credit. Debit = 1, Credit = 0. Debit half as likely as credit.
                 isDebit           = rbinom(n=dummyqty, size=1, prob=0.3333333333)
                 )

#get the calculated date fields...
TRANS_DATE_InDateFormat   <- as.Date("1970-01-01") + dummy_rows$TRANS_DATE
dummy_rows$TRANS_DATE_Mth <- month(TRANS_DATE_InDateFormat)
dummy_rows$TRANS_DATE_DoM <- day(TRANS_DATE_InDateFormat)
dummy_rows$TRANS_DATE_DoW <- wday(TRANS_DATE_InDateFormat)

dummy_rows$SYSTEM_DATE    <- dummy_rows$PostingDelayDays + dummy_rows$TRANS_DATE

SYSTEM_DATE_InDateFormat  <- as.Date("1970-01-01") + dummy_rows$SYSTEM_DATE
dummy_rows$TRANS_DATE_Mth <- month(SYSTEM_DATE_InDateFormat)
dummy_rows$TRANS_DATE_DoM <- day(SYSTEM_DATE_InDateFormat)
dummy_rows$TRANS_DATE_DoW <- wday(SYSTEM_DATE_InDateFormat)
```

```
#clear unused data
rm(TRANS_DATE_InDateFormat, SYSTEM_DATE_InDateFormat)
```

```
## Warning in rm(TRANS_DATE_InDateFormat, SYSTEM_DATE_InDateFormat): object
## 'TRANS_DATE_InDateFormat' not found

## Warning in rm(TRANS_DATE_InDateFormat, SYSTEM_DATE_InDateFormat): object
## 'SYSTEM_DATE_InDateFormat' not found
```

```
#create the flag to identify which records are random and which real
srcData_adj <- srcData_adj %>% mutate(isRandom = FALSE)

#We treat the IsRandom field as a logical field, we exclude it from the final model
Col_Spec[nrow(Col_Spec)+1,] <- list("isRandom", "logical", "exclude",
                                    as.numeric(NA), as.numeric(NA), FALSE, FALSE)

#append our dummy records to the real data
srcData_adj <- rbind(srcData_adj, dummy_rows)

#randomly sort
srcData_adj <- srcData_adj[sample(nrow(srcData_adj)),]
```

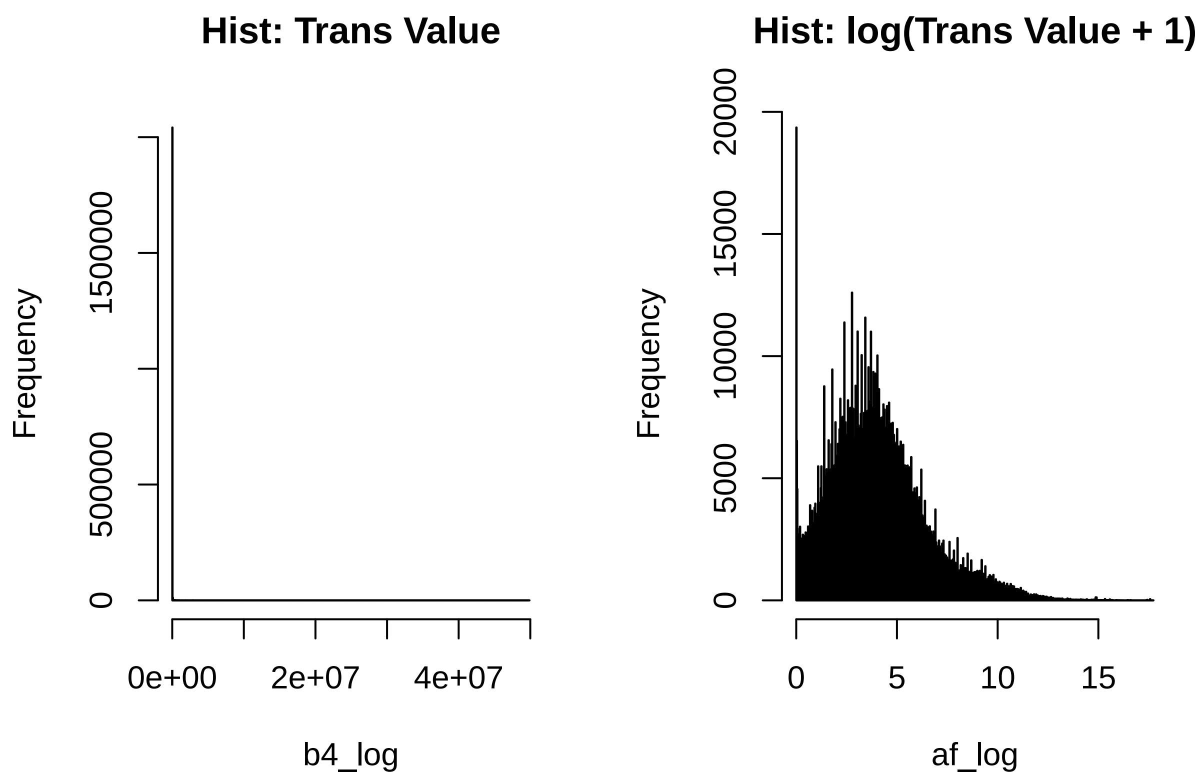
```
# take a copy for testing later
srcData_b4_ohot <- srcData_adj
```

Log of Transaction Value

The transaction value (BC_TRANS_VALUE) has a long tail, meaning most transactions are small and only a few are very large. Whereas deep learning works better with normally distributed data, so we apply the log(x+1) transform to transaction values (BC_TRANS_VALUE). This does not result in normally distributed data, but its a lot better than the raw data.

```
b4_log <- srcData_adj$BC_TRANS_VALUE
af_log <- log(srcData_adj$BC_TRANS_VALUE+1)
```

```
par(mfrow = c(1,2))
hist(b4_log, breaks = 1000, main="Hist: Trans Value")
hist(af_log, breaks = 1000, main="Hist: log(Trans Value + 1)")
```



Let's apply the log

```
# log the data, don't forget to add 1 because log(0)=inf.
srcData_adj$BC_TRANS_VALUE <- log(srcData_adj$BC_TRANS_VALUE + 1)

# record the fact that it has been logged
Col_Spec$Logged[which(Col_Spec$Name == "BC_TRANS_VALUE")] <- TRUE
```

Scale other Numerical Fields

Furthermore, deep learning works best when all the numerical (as opposed to categorical) data is at the same scale and with low values, preferably less than 1. At the same scale means subtracting the average and dividing by the standard deviation for each field. This also beings most of the data to be less than one.

However, the paper by PwC (Schreyer et al, 2018) reports good results with a different approach, presenting each figure as a proportion of the maximum for its field. This results in a value between 0 and 1 for all fields, this prioritises a 0 to 1 value over true scaling.

Note, after this process the Trans Value (BC_TRANS_VALUE) will have been both logged then 'minmax scaled'.

```r
# Scale the non-factor columns
# We save the min and max for each numerical column, so that we can 'unscale' later
for (i in 1:nrow(Col_Spec)) {

  if(Col_Spec$Type_final[i] == "numerical"){

    # get the relevant column's data
    col_num <- which(colnames(srcData_adj) == Col_Spec$Name[i])
    col_dat <- srcData_adj[,col_num]

    # record the min and max
    Col_Spec$Max[i] <- max(col_dat)
    Col_Spec$Min[i] <- min(col_dat)

    # minmax scale the data
    srcData_adj[,col_num] <- (col_dat - Col_Spec$Min[i]) / (Col_Spec$Max[i] - Col_Spec$Min[i])

    # record the fact that the data has been minmaxed
    Col_Spec$MinMaxed[i] <- TRUE

  } else {

    Col_Spec$Max[i]      <- NA
    Col_Spec$Min[i]      <- NA
    Col_Spec$MinMaxed[i] <- FALSE

  }

}
```

Convert Categorical Fields to One Hot

For feeding into a neural network, categorical fields need to be one-hot vectors. We will do this in R, using dplyr. BUT It would have been possible to have left this until entry to the model, in python for keras, using:

keras.utils.to_categorical(y, num_classes=None, dtype='float32')

```r
# convert each factor column to one-hot vectors, results in multipe fields per factor column
onehot <- function(dataframe, column_to_one_hot){

  #requires one column of unique values, column RowNum in this instance
  #also requires a column of 1's, column 'i' in this instance
```

```
  output <-  dataframe %>%
                select(eval(column_to_one_hot)) %>%
                  mutate(RowNum=row_number(), i=1) %>%
                    spread(key=eval(column_to_one_hot), value=i, fill=0) %>%
                      select(-RowNum)

  # append the col name to all new col names
  names(output) <- paste0(column_to_one_hot, "_", names(output))

  return(output)
}

TRANS_TYPE_ohot        <- onehot(srcData_adj, quote("TRANS_TYPE"))
OPERATOR_ohot          <- onehot(srcData_adj, quote("OPERATOR"))
DOCUMENT_REF_L1_ohot   <- onehot(srcData_adj, quote("DOCUMENT_REF_L1"))
CUST_REF_ohot          <- onehot(srcData_adj, quote("CUST_REF"))
SUPP_REF_ohot          <- onehot(srcData_adj, quote("SUPP_REF"))
ACCOUNT_NO_Digit1_ohot <- onehot(srcData_adj, quote("ACCOUNT_NO_Digit1"))
ACCOUNT_NO_Digit2_ohot <- onehot(srcData_adj, quote("ACCOUNT_NO_Digit2"))
ACCOUNT_NO_Digit3_ohot <- onehot(srcData_adj, quote("ACCOUNT_NO_Digit3"))
ACCOUNT_NO_Digit4_ohot <- onehot(srcData_adj, quote("ACCOUNT_NO_Digit4"))
ACCOUNT_NO_Digit5_ohot <- onehot(srcData_adj, quote("ACCOUNT_NO_Digit5"))
ACCOUNT_NO_Digit6_ohot <- onehot(srcData_adj, quote("ACCOUNT_NO_Digit6"))
ACCOUNT_NO_Digit7_ohot <- onehot(srcData_adj, quote("ACCOUNT_NO_Digit7"))

# Bring all columns together
# get list of non factor columns
col_refs <- which(!Col_Spec$Type_final %in% c("factor", "exclude"))
srcData_prepped <- cbind.data.frame(# not factor and not excluded
                                    srcData_adj[,col_refs],
                                    # factors
                                    TRANS_TYPE_ohot,
                                    OPERATOR_ohot,
                                    DOCUMENT_REF_L1_ohot,
                                    ACCOUNT_NO_Digit1_ohot,
                                    ACCOUNT_NO_Digit2_ohot,
                                    ACCOUNT_NO_Digit3_ohot,
                                    ACCOUNT_NO_Digit4_ohot,
                                    ACCOUNT_NO_Digit5_ohot,
                                    ACCOUNT_NO_Digit6_ohot,
                                    ACCOUNT_NO_Digit7_ohot,
                                    CUST_REF_ohot,
                                    SUPP_REF_ohot,
                                    # excluded from model
                                    DOCUMENT_REF = srcData_adj$DOCUMENT_REF,
                                    isRandom = srcData_adj$isRandom,
                                    stringsAsFactors = FALSE
                                    )

# record the order in which these columns appear
# we need this order for matrix operations later
Col_Spec_Sequence <- c( # not factor and not excluded
                        colnames(srcData_adj[,col_refs]),
```

```r
                        # factors
                        "TRANS_TYPE",
                        "OPERATOR",
                        "DOCUMENT_REF_L1",
                        "ACCOUNT_NO_Digit1",
                        "ACCOUNT_NO_Digit2",
                        "ACCOUNT_NO_Digit3",
                        "ACCOUNT_NO_Digit4",
                        "ACCOUNT_NO_Digit5",
                        "ACCOUNT_NO_Digit6",
                        "ACCOUNT_NO_Digit7",
                        "CUST_REF",
                        "SUPP_REF",
                        # excluded from model
                        "DOCUMENT_REF",
                        "isRandom")

Col_Spec_Sequence <- data.frame(Name = Col_Spec_Sequence,
                                Sequence = seq(from=1, to=length(Col_Spec_Sequence)),
                                stringsAsFactors = FALSE)

Col_Spec <- Col_Spec %>%
             left_join(Col_Spec_Sequence, by="Name") %>%
              arrange(Sequence)
```

```r
#Let's see how big this data is after we converted columns to one-hot...
paste0(format(object.size(srcData_prepped), units = "auto"))
```

```
## [1] "6 Gb"
```

A lot of data, mostly due to one hot encoding. Let's see the dimensions of the table, these will indicate the dims for the input layer of the autoencoder.

```r
dim(srcData_prepped)
```

```
## [1] 2068405     391
```

Each record is 393 columns wide, mostly due to one hot conversion and those fields will mostly be zeros. A very sparse matrix. The final two columns will not be submitted to the model, so input_dim = 391

Train, Validation and Test Sets

Autoencoders don't use x and y, aka input and labels. There is no 'y', 'label' or 'target'. The correct output for each record is simply the record itself. However, it is still useful to have a validation and test set. The train and test sets should be certain to include dummy data. The validation set will help us understand how model training is progressing, identifying overtraining. The Test set will help build confidence in how effective the model is.

```r
getdatasplit <- function(dataset, ProportionInSetA){

  ## Training set = proportion of the total data
  sample_size <- floor(ProportionInSetA * nrow(dataset))

  ## set the seed to make partition reproducible
  set.seed(20181212)
```

```r
  SetA_indices <- sample(seq_len(nrow(dataset)), size = sample_size, replace = F)

  SetA <- dataset[ SetA_indices, ]
  SetB <- dataset[-SetA_indices, ]

  return(list(SetA, SetB))
}

#first separate out the training set, exclude the isRandom field.
srcData_split <- getdatasplit(srcData_prepped, 0.67)
srcData_Train <- srcData_split[[1]]
srcData_TestAndValid  <- srcData_split[[2]]

#now separate out the validation and test sets
srcData_split <- getdatasplit(srcData_TestAndValid, 0.67)
srcData_Valid <- srcData_split[[1]]
srcData_Test  <- srcData_split[[2]]

rm(srcData_split, srcData_TestAndValid)

# report results

print(paste0("Rows in Training Set  : ",nrow(srcData_Train),
             '. As Propn:', round(nrow(srcData_Train)/nrow(srcData_prepped),2)))
print(paste0("Rows in Validation Set: ",nrow(srcData_Valid),
             '.  As Propn:', round(nrow(srcData_Valid)/nrow(srcData_prepped),2)))
print(paste0("Rows in Testing Set   : ",nrow(srcData_Test),
             '.  As Propn:', round(nrow(srcData_Test)/nrow(srcData_prepped),2)))

print(paste0("Dummy data in Training Set  : ", nrow(srcData_Train %>% filter(isRandom==TRUE))))
print(paste0("Dummy data in Validation Set: ", nrow(srcData_Valid %>% filter(isRandom==TRUE))))
print(paste0("Dummy data in Testing Set   : ", nrow(srcData_Test  %>% filter(isRandom==TRUE))))

# finally, remove the isRandom field and convert to matrix format for use in Keras.

srcData_Train_py <- as.matrix(srcData_Train %>% select (-c(isRandom, DOCUMENT_REF)))
srcData_Valid_py <- as.matrix(srcData_Valid %>% select (-c(isRandom, DOCUMENT_REF)))
srcData_Test_py  <- as.matrix(srcData_Test  %>% select (-c(isRandom, DOCUMENT_REF)))

# For the test set we will want to know the row references for the dummy records
srcData_Test_ID  <- srcData_Test %>%
                      mutate(ID = row_number()) %>%
                        select(ID, isRandom, DOCUMENT_REF)

#input_dim for model
input_dim = ncol(srcData_Train_py)
```

Transfer Data to Python

All fields except ‘isRandom’ and ‘Document_Ref’, are fed into the model

```
import pandas as pd
import numpy  as np
import os     as os
#dplyr for pandas in python
from   dfply import *
#convert to pandas
Train_py = r.srcData_Train_py
Testi_py = r.srcData_Test_py
Valid_py = r.srcData_Valid_py
input_dim= r.input_dim
```

Helper Function to Create Models

The model will be very similar to the PwC model version AE7:

1. 7 encoding layers (exc the input)
2. 1 bottleneck layer of 3 neurons (which means easy to plot data in 3D)
3. 7 decoding layers
4. Leaky ReLU activation, alpha =0.4
5. drop out at each layer, except the bottleneck

Here are the layer sizes: 391(Input)-256-128-64-32-16-8-4-3-4-8-16-32-64-128-188-256-391(Output)

A helper function has been written to create the model, with options to vary key components so we can grid search for the optimal model.

The code includes options for setting pretrained weights on the encoder. This will be used to view embeddings after we have trained the model. See F.Chollet on the approach to doing this: https://github.com/keras-team/keras/issues/41

If we had a single record and wanted to get activations (embeddings) for all layers in the model, then we could use an api like https://github.com/philipperemy/keract, but that is not our situation here. For embeddings we will want the activiatons at the bottleneck layer for all records.

```
from keras.layers import Input, Dense, LeakyReLU, Dropout, BatchNormalization, concatenate
from keras.models import Model
import numpy as np
#from keras import backend as K
def create_model_basic(
    input_dim          = input_dim,
    apply_dropout      = True,
    apply_batchnorm    = False,
    dropout_rate       = 0.2,
    leaky_alpha        = 0.4,
    # we may choose to see embeddings from pretrained models, rather than train from scratch...
    # for applying the pretrained weights of an existing model, we need encoding layer numbers
    # note, these may skip a layer between layers (ie 1,3,5 not 2,3),
    # this is because there may be a dropout layer between each dense layer
    get_embeddings     = False,
    pretrained_model   = None,
    # we have a dense layer and dropout layer for each encoding step.
    encode_layer_nodes = [256, 128, 64, 32, 16,   8,   4],
    # the bottleneck, or throat
    throat_layer_nodes = 3,
    # we need decode layers too
```

```
  decode_layer_nodes = [  4,   8, 16, 32, 64, 128, 256]):

#Kernel initialiser
kinit = 'glorot_normal'
#INPUT
the_input = Input(shape=(input_dim,))

####################################################################
# Encode Layers
####################################################################
encoded = the_input

for nodes in encode_layer_nodes:

  if(apply_dropout):
    encoded = Dropout(dropout_rate)(encoded)
  if(get_embeddings):
    # in the pretrained model layer[0] is the input layer
    # if we apply dropout then that's layer[1]
    # subsequently, this dense layer is layer[2] (or layer[1] without dropout)
    # and then the LeakyRelu is layer[3] (or layer[2] without dropout)
    lyr_idx = np.where(np.array(encode_layer_nodes) == nodes) # starts with 1
    lyr_idx = lyr_idx[0]*(2+apply_dropout+apply_batchnorm)+(1+apply_dropout)
    encoded = Dense(nodes, weights = pretrained_model.layers[lyr_idx[0]].get_weights())(encoded)
  else:
    encoded = Dense(nodes, kernel_initializer=kinit)(encoded)

  if(apply_batchnorm):
    encoded = BatchNormalization()(encoded)

  encoded = LeakyReLU(alpha=leaky_alpha)(encoded)

####################################################################
# Bottleneck (aka Throat)
####################################################################
# Typically 3 nodes for easy plotting of embeddings in 3D
# Note no droput into or out of bottleneck

encoded = Dense(throat_layer_nodes, kernel_initializer=kinit)(encoded)
encoded = LeakyReLU(alpha=leaky_alpha)(encoded)
####################################################################
# Decode Layers
####################################################################
decoded = encoded

for nodes in decode_layer_nodes:

  if(apply_dropout):
    decoded = Dropout(dropout_rate)(decoded)

  decoded = Dense(nodes, kernel_initializer=kinit)(decoded)

  if(apply_batchnorm):
```

```
    decoded = BatchNormalization()(decoded)

  decoded = LeakyReLU(alpha=leaky_alpha)(decoded)
#########################################################
# Reconstruct
#########################################################

#Split into
# the first 10 columns are all scaled values (input was mean=0, sd=1), no activation required
decode_TRANS_DATE        = Dense(1)(decoded) #intentionally no activation
decode_TRANS_DATE_Mth    = Dense(1)(decoded) #intentionally no activation
decode_TRANS_DATE_DoM    = Dense(1)(decoded) #intentionally no activation
decode_TRANS_DATE_DoW    = Dense(1)(decoded) #intentionally no activation
decode_SYSTEM_DATE       = Dense(1)(decoded) #intentionally no activation
decode_SYSTEM_DATE_Mth   = Dense(1)(decoded) #intentionally no activation
decode_SYSTEM_DATE_DoM   = Dense(1)(decoded) #intentionally no activation
decode_SYSTEM_DATE_DoW   = Dense(1)(decoded) #intentionally no activation
decode_PostingDelayDays  = Dense(1)(decoded) #intentionally no activation
decode_BC_TRANS_VALUE    = Dense(1)(decoded) #intentionally no activation

# this is a binary field, requires sigmoid activation
decode_isDebit           = Dense(1, activation='sigmoid')(decoded)

# the subsequent columns are all categories, requiring softmax activation
decode_TransType         = Dense( 18, kernel_initializer=kinit, activation='softmax')(decoded)
decode_OPERATOR          = Dense( 49, kernel_initializer=kinit, activation='softmax')(decoded)
decode_DOCUMENT_REF_L1   = Dense( 46, kernel_initializer=kinit, activation='softmax')(decoded)
decode_ACCOUNT_NO_Digit1 = Dense(  7, kernel_initializer=kinit, activation='softmax')(decoded)
decode_ACCOUNT_NO_Digit2 = Dense( 10, kernel_initializer=kinit, activation='softmax')(decoded)
decode_ACCOUNT_NO_Digit3 = Dense( 10, kernel_initializer=kinit, activation='softmax')(decoded)
decode_ACCOUNT_NO_Digit4 = Dense( 10, kernel_initializer=kinit, activation='softmax')(decoded)
decode_ACCOUNT_NO_Digit5 = Dense( 10, kernel_initializer=kinit, activation='softmax')(decoded)
decode_ACCOUNT_NO_Digit6 = Dense(  9, kernel_initializer=kinit, activation='softmax')(decoded)
decode_ACCOUNT_NO_Digit7 = Dense(  8, kernel_initializer=kinit, activation='softmax')(decoded)
decode_CUST_REF          = Dense(101, kernel_initializer=kinit, activation='softmax')(decoded)
decode_SUPP_REF          = Dense(100, kernel_initializer=kinit, activation='softmax')(decoded)

#########################################################
# Concatenate into one output
#########################################################

the_output = concatenate([decode_TRANS_DATE,
                          decode_TRANS_DATE_Mth,
                          decode_TRANS_DATE_DoM,
                          decode_TRANS_DATE_DoW,
                          decode_SYSTEM_DATE,
                          decode_SYSTEM_DATE_Mth,
                          decode_SYSTEM_DATE_DoM,
                          decode_SYSTEM_DATE_DoW,
                          decode_PostingDelayDays,
                          decode_BC_TRANS_VALUE,
                          decode_isDebit,
                          decode_TransType,
```

```
                              decode_OPERATOR,
                              decode_DOCUMENT_REF_L1,
                              decode_ACCOUNT_NO_Digit1,
                              decode_ACCOUNT_NO_Digit2,
                              decode_ACCOUNT_NO_Digit3,
                              decode_ACCOUNT_NO_Digit4,
                              decode_ACCOUNT_NO_Digit5,
                              decode_ACCOUNT_NO_Digit6,
                              decode_ACCOUNT_NO_Digit7,
                              decode_CUST_REF,
                              decode_SUPP_REF])
  if get_embeddings:
    #AUTOENCODER = ENCODE only
    model = Model(the_input, encoded)
  else:
    #AUTOENCODER = ENCODE + DECODE
    model = Model(the_input, the_output)

  return(model)
```

Instantiate the Model

The model creation method, above, allows many permutations, but for now we need to see an example, jus tto confirm it is reasonable. This example will be with dropout, and 7 layers We expect 391 inputs, which is the 393 in the data LESS the excluded fields; 'isRandom' , which would give the game away, and DOCUMENT_REF which adds no information to the model (unique record per row).

```
# create a baseline model
autoencoder_example = create_model_basic(input_dim      = input_dim,
                                         dropout_rate   = 0.0,
                                         apply_batchnorm= False)
```

```
autoencoder_example.summary()
```

```
## ________________________________________________________________________________________________
## Layer (type)                     Output Shape         Param #     Connected to
## ================================================================================================
## input_24 (InputLayer)            (None, 389)          0
## ________________________________________________________________________________________________
## dense_647 (Dense)                (None, 256)          99840       input_24[0][0]
## ________________________________________________________________________________________________
## leaky_re_lu_256 (LeakyReLU)      (None, 256)          0           dense_647[0][0]
## ________________________________________________________________________________________________
## dense_648 (Dense)                (None, 128)          32896       leaky_re_lu_256[0][0]
## ________________________________________________________________________________________________
## leaky_re_lu_257 (LeakyReLU)      (None, 128)          0           dense_648[0][0]
## ________________________________________________________________________________________________
## dense_649 (Dense)                (None, 64)           8256        leaky_re_lu_257[0][0]
## ________________________________________________________________________________________________
## leaky_re_lu_258 (LeakyReLU)      (None, 64)           0           dense_649[0][0]
## ________________________________________________________________________________________________
## dense_650 (Dense)                (None, 32)           2080        leaky_re_lu_258[0][0]
```

```
## ____________________________________________________________________________________________________
## leaky_re_lu_259 (LeakyReLU)     (None, 32)          0           dense_650[0][0]
## ____________________________________________________________________________________________________
## dense_651 (Dense)               (None, 16)          528         leaky_re_lu_259[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_260 (LeakyReLU)     (None, 16)          0           dense_651[0][0]
## ____________________________________________________________________________________________________
## dense_652 (Dense)               (None, 8)           136         leaky_re_lu_260[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_261 (LeakyReLU)     (None, 8)           0           dense_652[0][0]
## ____________________________________________________________________________________________________
## dense_653 (Dense)               (None, 4)           36          leaky_re_lu_261[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_262 (LeakyReLU)     (None, 4)           0           dense_653[0][0]
## ____________________________________________________________________________________________________
## dense_654 (Dense)               (None, 3)           15          leaky_re_lu_262[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_263 (LeakyReLU)     (None, 3)           0           dense_654[0][0]
## ____________________________________________________________________________________________________
## dense_655 (Dense)               (None, 4)           16          leaky_re_lu_263[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_264 (LeakyReLU)     (None, 4)           0           dense_655[0][0]
## ____________________________________________________________________________________________________
## dense_656 (Dense)               (None, 8)           40          leaky_re_lu_264[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_265 (LeakyReLU)     (None, 8)           0           dense_656[0][0]
## ____________________________________________________________________________________________________
## dense_657 (Dense)               (None, 16)          144         leaky_re_lu_265[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_266 (LeakyReLU)     (None, 16)          0           dense_657[0][0]
## ____________________________________________________________________________________________________
## dense_658 (Dense)               (None, 32)          544         leaky_re_lu_266[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_267 (LeakyReLU)     (None, 32)          0           dense_658[0][0]
## ____________________________________________________________________________________________________
## dense_659 (Dense)               (None, 64)          2112        leaky_re_lu_267[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_268 (LeakyReLU)     (None, 64)          0           dense_659[0][0]
## ____________________________________________________________________________________________________
## dense_660 (Dense)               (None, 128)         8320        leaky_re_lu_268[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_269 (LeakyReLU)     (None, 128)         0           dense_660[0][0]
## ____________________________________________________________________________________________________
## dense_661 (Dense)               (None, 256)         33024       leaky_re_lu_269[0][0]
## ____________________________________________________________________________________________________
## leaky_re_lu_270 (LeakyReLU)     (None, 256)         0           dense_661[0][0]
## ____________________________________________________________________________________________________
## dense_662 (Dense)               (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_663 (Dense)               (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_664 (Dense)               (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_665 (Dense)               (None, 1)           257         leaky_re_lu_270[0][0]
```

```
## ____________________________________________________________________________________________________
## dense_666 (Dense)                (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_667 (Dense)                (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_668 (Dense)                (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_669 (Dense)                (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_670 (Dense)                (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_671 (Dense)                (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_672 (Dense)                (None, 1)           257         leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_673 (Dense)                (None, 18)          4626        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_674 (Dense)                (None, 49)          12593       leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_675 (Dense)                (None, 46)          11822       leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_676 (Dense)                (None, 7)           1799        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_677 (Dense)                (None, 10)          2570        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_678 (Dense)                (None, 10)          2570        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_679 (Dense)                (None, 10)          2570        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_680 (Dense)                (None, 10)          2570        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_681 (Dense)                (None, 9)           2313        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_682 (Dense)                (None, 8)           2056        leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_683 (Dense)                (None, 101)         25957       leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## dense_684 (Dense)                (None, 100)         25700       leaky_re_lu_270[0][0]
## ____________________________________________________________________________________________________
## concatenate_18 (Concatenate)     (None, 389)         0           dense_662[0][0]
##                                                                  dense_663[0][0]
##                                                                  dense_664[0][0]
##                                                                  dense_665[0][0]
##                                                                  dense_666[0][0]
##                                                                  dense_667[0][0]
##                                                                  dense_668[0][0]
##                                                                  dense_669[0][0]
##                                                                  dense_670[0][0]
##                                                                  dense_671[0][0]
##                                                                  dense_672[0][0]
##                                                                  dense_673[0][0]
##                                                                  dense_674[0][0]
##                                                                  dense_675[0][0]
##                                                                  dense_676[0][0]
```

```
##                                                                  dense_677[0][0]
##                                                                  dense_678[0][0]
##                                                                  dense_679[0][0]
##                                                                  dense_680[0][0]
##                                                                  dense_681[0][0]
##                                                                  dense_682[0][0]
##                                                                  dense_683[0][0]
##                                                                  dense_684[0][0]
## ====================================================================================================
## Total params: 287,960
## Trainable params: 287,960
## Non-trainable params: 0
## ____________________________________________________________________________________________________
```

290k parameters to train, should be feasible with 1.3million records.

Helper Function to Compile and Fit Models

We will do a grid scan of the model options such as: batch size [64, 256, 2048] dropout rate [0.0, 0.1, 0.2]

We could also scan options such as layer quantity or node quantity within each layer, but time is limited. At the heart of the grid scan will be a method to create the model, compile it, then fit the data. This function may choose to use tensorflow callbacks (tensorflow is the backend to Keras on this rig)

```
from keras import callbacks
# hook up to Tensorboard for keeping records of progress
# to use the Tensorboard, go to command line, set cd "current project directory", then:
# tensorboard --logdir GraphRMS
# then
# http://localhost:6006/
## Tensorboard callback
tbCallback_ae_basic = callbacks.TensorBoard(log_dir        = './Tensorboard_ae_basic',
                                            histogram_freq = 0,
                                            write_graph    = True)
## Checkpoint callback
cpCallback_ae_basic = callbacks.ModelCheckpoint('./Checkpoints/ae_basic_{epoch:02d}.hdf5',
                                                monitor='val_acc', verbose=1,
                                                save_best_only=True, mode='max')

##Earlystopping, stop training when validation accuracy ceases to improve
esCallback          = callbacks.EarlyStopping(monitor   = 'val_loss', min_delta = 0.00005,
                                              patience = 200,        verbose   = 0,
                                              mode     = 'auto',     baseline  = None)
```

```
os.chdir('c:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM\\AcctgAnomalyDe
def create_compile_fit(input_dim       = 391,
                       dropout_rate    = 0.1,
                       apply_batchnorm = True,
                       batch_size      = 2048,
                       epochs          = 500,
                       Train_data      = Train_py,
                       Valid_data      = Valid_py,
                       callbacks       = None):
  ## Create model
```

```
  if(dropout_rate == 0):
    apply_dropout = False
  else:
    apply_dropout = True

  model = create_model_basic(input_dim      = input_dim,
                             apply_dropout  = apply_dropout,
                             dropout_rate   = dropout_rate,
                             apply_batchnorm= apply_batchnorm)
  ## Compile Model
  model.compile(optimizer = 'adadelta',
                loss      = 'binary_crossentropy'
                # metrics   = ['accuracy']
                )

  ## Fit model
  model.fit(x          = Train_data,
            # note for autoencoders the x is the same as the y, output=input
            y          = Train_data,
            epochs     = epochs,
            batch_size = batch_size,
            shuffle    = True,
            validation_data = (Valid_data, Valid_data),
            callbacks  = callbacks)
  return model
```

Helper Function to Save Results

As we're creating a number of models we will need to save them to file.

```
def save_model(keras_model, filename):
  import numpy as np
  #set working directory
  os.chdir('c:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM\\AcctgAnomaly

  # save model
  keras_model.save(filename+'.h5')

  # save training history
  #import pickle
  #with open('./SaveModels/autoencoder_basic_history.pickle', 'wb') as file_pi:
  #    pickle.dump(autoencoder_basic.history.history, file_pi)

  # save training history
  l_loss  = np.array(keras_model.history.history['loss'])
  l_loss  = np.reshape(l_loss, (l_loss.shape[0],1))
  l_vloss = np.array(keras_model.history.history['val_loss'])
  l_vloss = np.reshape(l_vloss, (l_vloss.shape[0],1))

  np.save(file = filename+'_history',
          arr  = np.concatenate((l_loss, l_vloss), axis=1))
```

Grid Scan the Model Parameters

All data is now either binary (precisely 1 or 0) or numerical (between 1 and 0). So, we can use 'binary_crossentropy' for our loss calculation, as per the PwC paper. If the numerical fields had simply been scaled, with some values greater than 1, then we would have been tempted to use 'mean_squared_error', but this would not have suited the categorical fields. This huighlights an advantage of the minmax scaling, it enables us to use one method of loss calculation for both categorical and numerical fields.

The following hyper parameters remain to be identified: Optimal batch size Whether regularisation via drop out is helpful Whether regularisation via batch normalisation is helpful

A grid scan of the various hyper param permutations can be carried out to find the optimal arrangement. We simply train the various instances of the model and see which performs best

```
import pandas as pd
from keras import callbacks
#dplyr for pandas in python
from dfply import *
# set FP16, else training is far too slow for grid search (24hrs per model, even with RTX2070)
#from keras.backend.common import set_floatx
#set_floatx('float16')
os.chdir('c:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM\\AcctgAnomalyDe
params =  {'dropout_rate'   : [  0.0, 0.1],
           'batch_size'     : [  256, 1024, 2048],
           'batch_norm'     : [False, True]
           }
#prep results table
results = None
results = pd.DataFrame(columns=['dropout_rate', 'batch_size', 'batch_norm',
                                'val_loss', 'loss', 'epoch'])

loop_count = -1
for dropout_rate in params.get('dropout_rate'):
  for batch_size in params.get('batch_size'):
    for batch_norm in params.get('batch_norm'):

      loop_count += 1
      params_current = {'dropout_rate' : dropout_rate,
                        'batch_size'   : batch_size}

      # filename for model instance
      filename = ('DropoutRate_'+ str(dropout_rate) +
                  '_BatchSize_' + str(batch_size) +
                  '_BatchNorm_' + str(batch_norm))

      # create csv logger callback
      csvCallback = callbacks.CSVLogger(filename+'_Log.log')

      # create model
      model_instance = create_compile_fit(input_dim       = input_dim,
                                          dropout_rate    = dropout_rate,
                                          apply_batchnorm = batch_norm,
                                          batch_size      = batch_size,
                                          epochs          = 400,
                                          Train_data      = Train_py,
```

```
                                          Valid_data      = Valid_py,
                                          callbacks       = [csvCallback])
      # save the model
      save_model(model_instance, filename)

      # get result, i.e. min validation error and train error for same epoch
      best_valid      = min(model_instance.history.history['val_loss'])
      best_valid_epc  = np.where(model_instance.history.history['val_loss'] == best_valid)
      best_valid_epc  = best_valid_epc[0].item() #tie breaker, take first. need item else rtn array
      matching_train  = model_instance.history.history['loss'][best_valid_epc]

      # save results to table for comparison
      results.loc[len(results)] = [dropout_rate, batch_size, batch_norm,
                                   best_valid, matching_train, best_valid_epc]

      print("loop: ", loop_count)
      print('\n')
      print(results.loc[len(results)-1])
      print('\n')

# save to file because this took a long time (approx 10hrs) to complete!
results.to_csv('GridScan_Results.csv')
```

Read Grid Scan Results

```
# load from file because above hyper param optimisation with 2000 epochs per model takes ~24hrs
import pandas as pd
from dfply import *
import os as os
os.chdir('c:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM\\AcctgAnomalyDe
results = pd.read_csv('GridScan_Results.csv')
pd.set_option("display.max_columns",12)
print(results >> arrange(X.val_loss, ascending=True))
```

```
##    dropout_rate  batch_size  batch_norm  val_loss      loss  epoch
## 0           0.0         256       False  0.016406  0.017909    391
## 1           0.0        1024       False  0.017092  0.019047    396
## 2           0.0         256        True  0.018828  0.017018    292
## 3           0.0        2048       False  0.019154  0.020428    387
## 4           0.1         256        True  0.053829  0.052933     25
## 5           0.1        2048       False  0.062139  0.063302     48
## 6           0.1        1024       False  0.063732  0.065805     16
## 7           0.1         256       False  0.063739  0.066403      3
```

From the above table we see the lowest validation loss can be found on the model with 256 batch size, no dropout and no batch normalisation. But, that model returns a validation loss lower than the training loss. We should be careful with this result.

Let's see how this model performs for separating dummy records from real records.

Load Selected Model From File

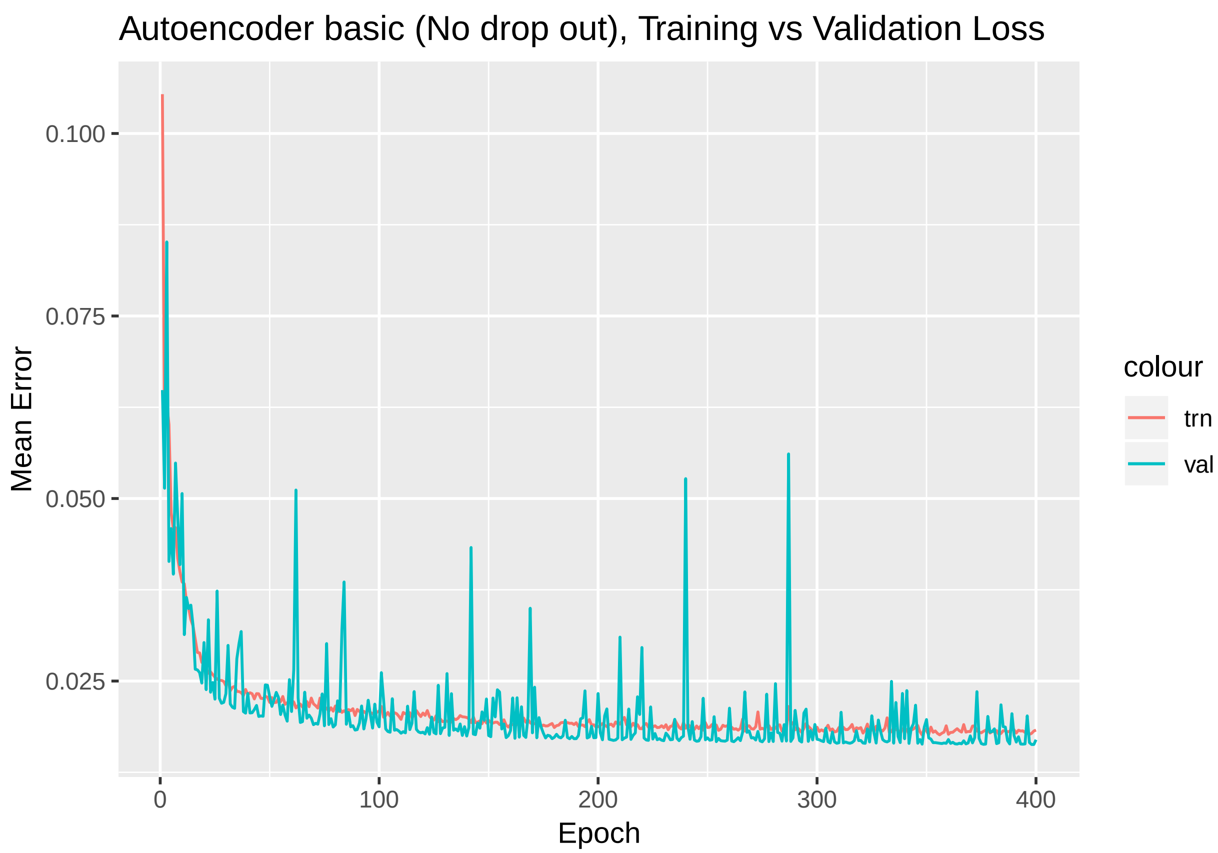
```
from keras.models import load_model
from os import path, chdir
import pickle
def get_model_n_history_file(filename):
  chdir('c:\\Users\\User\\OneDrive\\Oliver\\0_OM\\Training\\DeepLearning_Udacity\\LSTM\\AcctgAnomalyDete
  the_model_locn   = path.join('.\\SaveModels', filename + ".h5")
  the_model        = load_model(the_model_locn)
  the_history_locn = path.join('.\\SaveModels', filename + "_history.npy")
  the_history      = np.load(file = the_history_locn)

  return(the_model, the_history)
```

```
autoencoder_basic, history_autoencoder_basic = get_model_n_history_file(
  filename = "DropoutRate_0.0_BatchSize_256")
```

Plot Convergence

```
library(ggplot2)
library(gridExtra)

plot_convergence <- function(python_history_object, title_type){

  #example of python history object = py$history_autoencoder_basic

  require(ggplot2)
  require(gridExtra)

  history_ae_basic_train <- unlist(python_history_object[,1])
  history_ae_basic_valid <- unlist(python_history_object[,2])
  history_ae_basic_epoch <- seq_along(history_ae_basic_train)

  history_ae_basic <- cbind.data.frame(epoch    = history_ae_basic_epoch,
                                       loss     = history_ae_basic_train,
                                       val_loss = history_ae_basic_valid)

  p <- ggplot(data = history_ae_basic, aes(x=epoch))    +
       geom_line(aes(y = loss,     col = "trn"))    +
       geom_line(aes(y = val_loss, col = "val")) +
       xlab("Epoch")                                  +
       ylab("Mean Error") + #ylim(0.6,1.4)+
       ggtitle(paste0("Autoencoder basic (",title_type,"), Training vs Validation Loss"))

  grid.arrange(p, ncol=1)
}
```

```
plot_convergence(python_history_object = py$history_autoencoder_basic, title_type="No drop out")
```



Get Re-Constructions of Every Test Record

We are hoping that the average reconstruction error is greater for randomly created dummy records, of which there are ~700 for training, than it is for real records, of which there are 1.5 million for training.

The first step is to create a reconstruction of every record in the training table. We can later compare these with the original records.

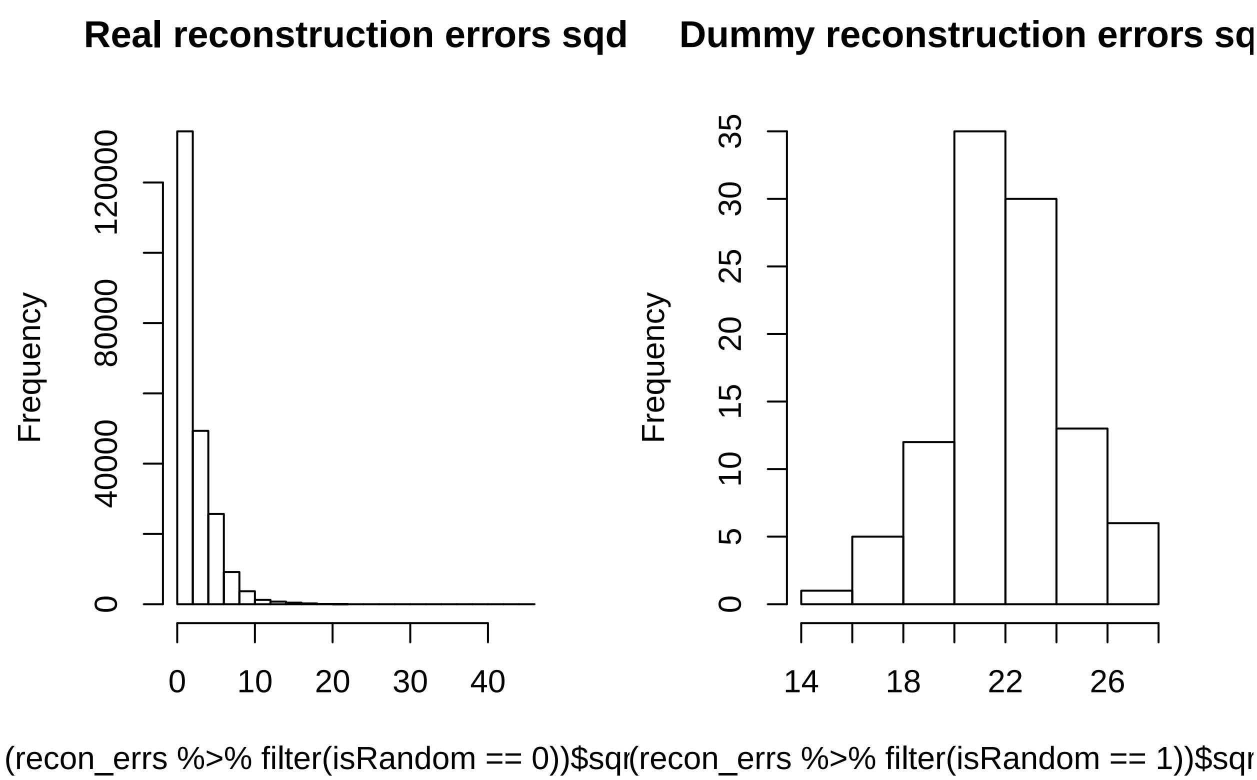
```
Testi_preds = autoencoder_basic.predict(Testi_py)
```

Explore Distribution of Errors on the Test Set

```
get_reconstruction_errors <- function(py_dat_in, py_dat_out, srcData_Test_ID){

  #example of py_dat_in py$Testi_py
  #example of py_dat_out py$Testi_preds

  # Get reconstruction errors for each record
  recon_errs <- cbind.data.frame(error    = rowSums(py_dat_in - py_dat_out),
                                 sqrd_err = rowSums((py_dat_in - py_dat_out)^2),
                                 isRandom = srcData_Test_ID$isRandom, # 1=dummy, 0=real
                                 DOCUMENT_REF = srcData_Test_ID$DOCUMENT_REF,
                                 stringsAsFactors = FALSE
```

```r
                                  )

  # Calculate mean squared error of the real (not dummy) data. Sum of errors div by qty
  mean_sqrd_err_real  <- sum(recon_errs$sqrd_err * !recon_errs$isRandom) / sum(!recon_errs$isRandom)

  # calculate the mean squared error of the dummy data. Sum of errors div by qty
  mean_sqrd_err_dummy <- sum(recon_errs$sqrd_err *  recon_errs$isRandom) / sum( recon_errs$isRandom)

  # Plot distribution of errors, real vs dummy data

  ## first for real data
  par(mfrow=c(1,2))
  hist((recon_errs %>% filter(isRandom == 0))$sqrd_err, main="Real reconstruction errors sqd")
  #qqnorm((recon_errs %>% filter(isRandom == 0))$sqrd_err)

  ## now for dummy data
  hist((recon_errs %>% filter(isRandom == 1))$sqrd_err, main="Dummy reconstruction errors sqd")
  #qqnorm((recon_errs %>% filter(isRandom == 1))$sqrd_err)

  return(list(mean_sqrd_err_real, mean_sqrd_err_dummy, recon_errs))
}
```

```r
mean_sqrd_errs <- get_reconstruction_errors(py_dat_in  = py$Testi_py,
                                            py_dat_out = py$Testi_preds,
                                            srcData_Test_ID = srcData_Test_ID)
```



```r
mean_sqrd_err_real  <- mean_sqrd_errs[[1]]
mean_sqrd_err_dummy <- mean_sqrd_errs[[2]]
recon_errs          <- mean_sqrd_errs[[3]]
rm(mean_sqrd_errs)
```

The above charts show that the dummy reconstruction errors are far larger than the real data, the model appears to be sifting dummy records from real records very well.

Let's see the reconstruction errors of 5 real records vs 5 dummy records

```r
kable(rbind(sample_n(recon_errs, 5) %>% select(isRandom, sqrd_err),
                 recon_errs %>% filter(isRandom==1) %>% sample_n(5) %>% select(isRandom, sqrd_err)
      ))
```

	isRandom	sqrd_err
133854	FALSE	1.5848260
81137	FALSE	6.5572770
34886	FALSE	0.3898606
90299	FALSE	0.9337914
92227	FALSE	1.1401269
2	TRUE	21.4905494
80	TRUE	21.9330628
15	TRUE	22.1732879
68	TRUE	22.0041937
20	TRUE	21.7365755

This is somewhat heartening, the dummy data has a very different distribution of errors to the real data. But we have a highly skewed data set, only 70+ dummy records being compared with hundreds of thousands of test records. For what its worth with only 100+ dummy records, we can do a formal t.test to confirm the significance of the difference in distributions.

```r
#Hypothesis test, are these significantly different distributions of correlation values?
ttest_result <- t.test(error ~ isRandom, paired = FALSE, var.equal=FALSE, data=recon_errs)

print(paste0("Dummy data significantly different to real data ?",
             cat("\n"),
             " The T Test's p-value = ",
             round(ttest_result$p.value,8),
             " . If <0.001 then significantly different (@99.9% confidence)"))
```

```
##
## [1] "Dummy data significantly different to real data ? The T Test's p-value = 0 . If <0.001 then sign
```

Thats fairly conclusive, the reconstruction errors of dummy records are significantly different from the real records (>99.9% confident).

Let's summarise our measures of the error differences.

```r
reconstruction_errs <- cbind.data.frame(record_type   = c("Real", "Dummy"),
                                        qty_records   = c(sum(!srcData_Test_ID[,2]),
                                                          sum( srcData_Test_ID[,2])),
                                        mean_sqrd_err = c(mean_sqrd_err_real,mean_sqrd_err_dummy),
                                        stringsAsFactors = FALSE
                                        )
kable(reconstruction_errs)
```

record_type	qty_records	mean_sqrd_err
Real	225148	2.25534
Dummy	102	22.00035

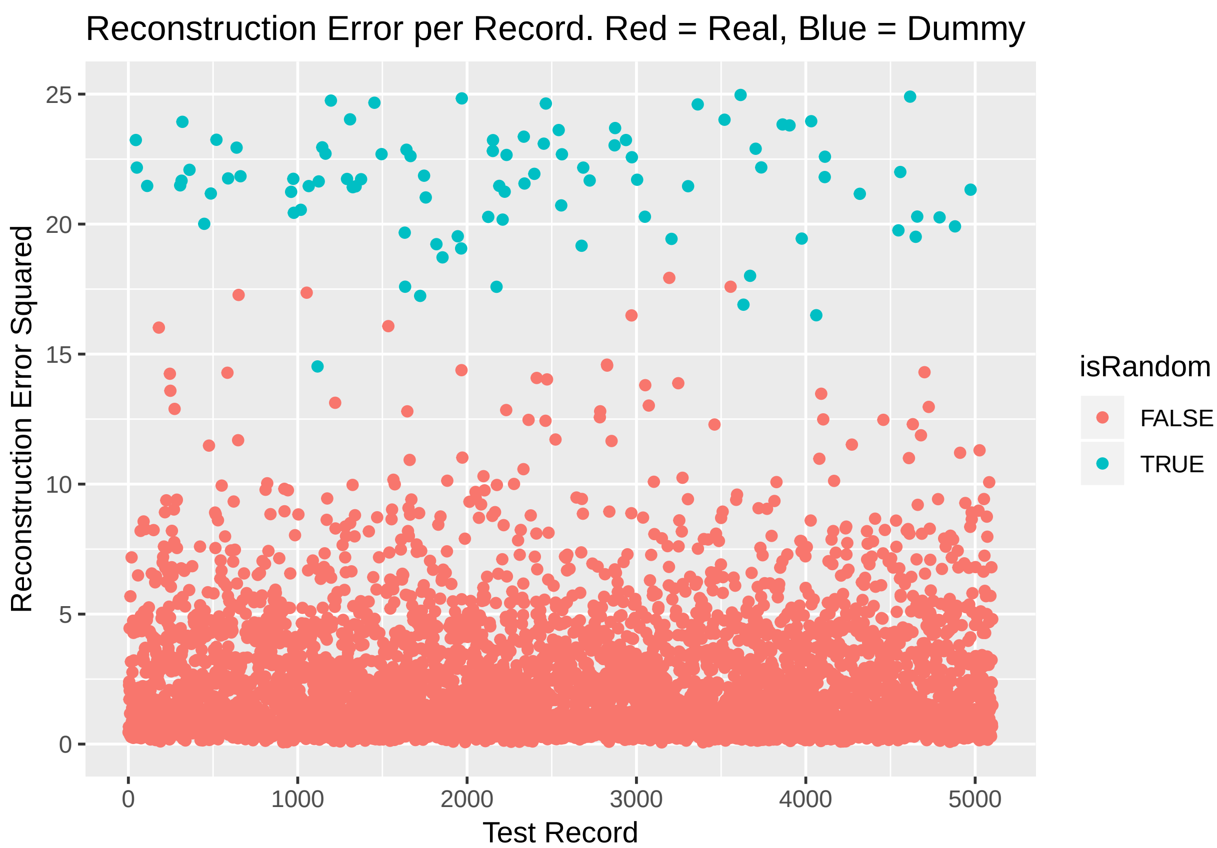
```
#print(reconstruction_errs)
```

Scatterplot of errors on real data vs errors on dummy data

PwC presented their errors in this format, great for getting a feel of the differences in errors. Requires we randomly select 5000 real records, as there are too many (250k) to plot.

```
library(ggplot2)
errors_to_plot <- rbind(sample_n(recon_errs, 5000) %>% select(isRandom, sqrd_err),
                        recon_errs %>% filter(isRandom==1) %>% select(isRandom, sqrd_err))
errors_to_plot <- errors_to_plot[sample(nrow(errors_to_plot)),]


m <- ggplot(data = errors_to_plot, aes(x = seq_along(isRandom),
                                      y = sqrd_err,
                                      color = isRandom)) +
            geom_point() +
            ylim(low=0,high=25) +
            #geom_smooth(method = "lm", se=TRUE, color="black", formula = y ~ x) +
            xlab("Test Record") +
            ylab("Reconstruction Error Squared") +
            #annotate("text", x = 20, y = -0.4, label = label, parse = F)+
            ggtitle("Reconstruction Error per Record. Red = Real, Blue = Dummy")
m
```

```
## Warning: Removed 10 rows containing missing values (geom_point).
```



Inspect Examples

Let's inspect randomly sampled examples of real, original (not reconstructed) data where the reconstruction error was large. These are candidates for erroneous or otherwise unusual transactions. Will their unusual nature be clear to a human mind.

Sample Candidates

Let's inspect the top 100 transactions with the largest reconstruction errors which were not dummy records.

```
# find top 1000th quantile
#toperrors_rng <- quantile(recon_errs$sqrd_err, probs = c(0.999, 1))

# create sample within that quantile
#the_sample <- sample_n(recon_errs %>% filter(sqrd_err > toperrors_rng[1] & isRandom == 0),
#                         size = 5,
#                         replace = FALSE)
# get candidates
#candidates <- (srcData_Test %>% select(-c(isRandom, DOCUMENT_REF)))[as.numeric(rownames(the_sample)),]
#candidate_row_refs <- rownames(candidates)
candidate_row_refs <- rownames(recon_errs %>%
                                 arrange(desc(sqrd_err)) %>%
                                   filter(isRandom == 0))[1:100]
```

```r
candidates          <- (srcData_Test
                        %>% select(-c(isRandom, DOCUMENT_REF)))[as.numeric(candidate_row_refs ),]
```

Inspect examples

To make sense of what we see, we'll need to 'un-minmax' the data, un-log the transaction value and return one_hot vectors to category labels we can easily read. This needs a fair bot of code...

```r
unscale_unhot <- function(data_mat,        # data to be unscaled and unhotted, in matrix (not df)
                          data_af_ohot,    # example of data frame to match input (cols for matrix)
                          Column_Spec  = Col_Spec,    # column specifications
                          apply_levels = TRUE,        # whether to convert levels to chars
                          factorlevels = factorlevels,# levels to be used for chars
                          apply_dateformat = TRUE,    # whether to convert date decimals to dates
                          apply_isRandom   = TRUE,    # whether to add the isRandom field
                          apply_docref     = TRUE     # whether to add the document reference
                          ) {

  ################################################################################
  # TestData
  ################################################################################

  #data_mat         <- as.matrix(srcData_prepped[1:100,] %>% select(-isRandom, -DOCUMENT_REF))
  #data_af_ohot     <- srcData_prepped
  #Column_Spec      <- Col_Spec
  #apply_levels     <- TRUE
  #apply_dateformat <- TRUE
  #apply_isRandom   <- TRUE
  #apply_docref     <- TRUE

  ################################################################################
  # Preparation
  ################################################################################

  ## Convert input matrix to data frame with appropriate column names
  data_df <- as.data.frame(data_mat)

  # get list of columns which we want (ie, not excluded, eg isRandom)
  Excluded_Cols <- (Column_Spec %>% filter(Type_final == "exclude"))$Name
  Included_Cols <- which(!colnames(data_af_ohot) %in% Excluded_Cols)

  # apply the column names in the correct order
  colnames(data_df) <- colnames(data_af_ohot)[Included_Cols]

  # get a record of those input dims and column name, we create a mapping with them later
  input_dims <- rbind.data.frame(Sequence = seq(from=1, to=length(colnames(data_df))))
  colnames(input_dims) <- colnames(data_df)

  ## Special rules to handle the columns excluded form the model; isRandom and DocumentRef
  # separate out isRandom and DocumentRef using the row references,
  # row references are row names on the data frame

  MissingCols <- data_af_ohot %>%
```

```
                 tibble::rownames_to_column() %>%
                   select_(.dots = Excluded_Cols, "rowname")

MissingCols <- data_df  %>%
                   tibble::rownames_to_column() %>%
                     inner_join(MissingCols, by="rowname") %>%
                       select_(.dots = Excluded_Cols, "rowname")

# remove the 'excluded columns' from data_af_hot
data_af_ohot <- data_af_ohot[,Included_Cols]

################################################################################
# PART 1, UN ONE HOT
################################################################################

# get list of factor columns in original order
onehot_col_names <- (Col_Spec %>% filter(Type_final == "factor") %>% arrange(Sequence))$Name

# get quantity of one hot columns associated with each column name
onehot_col_qtys  <- sapply(onehot_col_names, function(x) ncol(data_df %>% select(starts_with(x))))

# record these results in a table
onehot_cols <- cbind.data.frame(Name = onehot_col_names,
                                Qty  = onehot_col_qtys,
                                stringsAsFactors = F)

# for each factor we use matrix multiplication and rowsums to un-onehot.
for (i in 1:nrow(onehot_cols)) {

  # create mask which gives levels of factor (not values of factor)
  mask_per_row <- seq(from = 1, to = onehot_cols[i,2], by = 1)

  # select the columns to reverse from onehot
  data_to_mult <- as.matrix(data_df %>% select(starts_with(onehot_cols[i,1])))

  #round all values, we need integers
  data_to_mult <- round(data_to_mult,0)

  # multiply the levels-mask by the one-hot-vectors
  resulting_levels <- t(t(data_to_mult) * mask_per_row)
  resulting_levels <- rowSums(resulting_levels)

  #strip dimnames
  attributes(resulting_levels)$names <- NULL

  # bind to other results
  if(i==1){
    resulting_levels_all <- resulting_levels
  } else {
    resulting_levels_all <- cbind(resulting_levels_all, resulting_levels)
  }

}
```

```r
#apply column names
colnames(resulting_levels_all) <- onehot_cols$Name

################################################################################
# PART 2, UN MINMAX SCALE
################################################################################

# get list of minmaxed columns in original order
minmax_cols <- Column_Spec %>% filter(MinMaxed==TRUE) %>% arrange(Sequence)

# filter down to data in same order
data_df_minmax <- data_df %>% select_(.dots = minmax_cols$Name)

#prepare min and max values, in same order, of course
min_values <- minmax_cols$Min
max_values <- minmax_cols$Max

# Multiply by Range size (Max-Min)
## We could use other approaches, but matrix multiplication is fastest for millions of rows
data_mat_minmax<- t(t(as.matrix(data_df_minmax)) * (max_values - min_values))

# Add min
data_mat_minmax<- t(t(data_mat_minmax) + min_values)

################################################################################
# PART 3 combine the results (factor, non-factor, binary (ie isDebit))
################################################################################

output <- cbind.data.frame(resulting_levels_all,
                           data_mat_minmax,
                           isDebit = data_df$isDebit,
                           stringsAsFactors = FALSE)

#convert to data frame and reorder columns so same as original data
output_colorder <- (Col_Spec %>%
                      filter(Type_final != "exclude") %>%
                        arrange(Sequence)
                    )$Name

output <- output %>%
            select_(.dots = output_colorder)

# round all non-factor values EXCEPT BC_TRANS_VALUE, which is rounded to 2 places
# first get list of affected columns
rounding_cols <- (Col_Spec %>%
                    filter(!Type_final %in% c("exclude", "factor")
                           &
                           !Name == "BC_TRANS_VALUE")
                  )$Name

output <- output %>% mutate_at(vars(rounding_cols), funs(round(., 0)))

# now round BC_TRANS_VALUE to 2 decimal places (ie pennies)
```

```r
output <- output %>% mutate_at("BC_TRANS_VALUE", funs(round(., 2)))

# get a record of those output dims and column name, we create a mapping with them later
output_dims <- rbind.data.frame(Sequence = seq(from=1, to=length(colnames(output))))
colnames(output_dims) <- colnames(output)

#################################################################################
# PART 4. If desired, reverse factor levels back to original data
#################################################################################

if(apply_levels){

  #get reference to factor columns
  output_factor_cols <- which(colnames(output) %in% onehot_col_names)

  for (j in 1:length(output_factor_cols)){

    # get factorlevels for the current column name
    current_col_name   <- colnames(output)[output_factor_cols[j]]
    current_col_ref    <- which(names(factorlevels) == current_col_name)
    current_col_levels <- factorlevels[[current_col_ref]]

    # apply levels
    output[, output_factor_cols[j]] <- current_col_levels[output[, output_factor_cols[j]]]

    # apply original column type, necessary for integers which may be stored as chars
    current_col_spec<- Column_Spec %>% filter(Name == current_col_name)

    if(current_col_spec$Type_init[1] == "integer"){
      output[, output_factor_cols[j]] <- as.numeric(output[, output_factor_cols[j]])
    }
  }
}

#################################################################################
# PART 5. apply date format and missing columns (isRandom, DocumentRef)
#################################################################################

if(apply_dateformat){

  for (j in 1:ncol(output)){

    #get the column type
    date_colname <- colnames(output)[j]
    date_colspec <- Column_Spec %>% filter(Name == date_colname)

    if(date_colspec$Type_init[1] == "Date"){
      output[, j] <- as.Date(output[, j] , origin="1970-01-01")
    }
  }
}

#apply the rownames
```

```
  rownames(output) <- rownames(data_mat)

  #apply isRandom
  if(apply_isRandom){
    output$isRandom <- (output %>%
                          tibble::rownames_to_column() %>%
                           inner_join(MissingCols, by="rowname"))$isRandom

  }

  #apply the DocumentRef so we can easily inspect source documents during testing
  if(apply_docref){
    output$DOCUMENT_REF <- (output %>%
                              tibble::rownames_to_column() %>%
                               inner_join(MissingCols, by="rowname"))$DOCUMENT_REF
  }

  ################################################################################
  # PART 6. Build column mapping, input dims to output dims
  ################################################################################
  mapping <- input_dims

  for(i in 1:ncol(output_dims)){

    # get the current output name and column sequence
    output_name <- colnames(output_dims)[i]
    output_seq  <- (output_dims %>% select_(.dots=output_name))[1,1]

    #is output column a factor?
    Type_final <- (Column_Spec %>% filter(Name == output_name))$Type_final[1]

    if(Type_final != "factor"){
      #if not a factor, then simply get the one value for the sequence
      colrefs <- input_dims %>% select_(.dots = output_name)
    }else{
      #if it is a factor, then find input names which start with the output name
      colrefs <- input_dims %>% select(starts_with(output_name))
    }

    # record result in mapping table
    colrefs <- as.matrix(colrefs)[1,]
    mapping[2,colrefs] <- output_seq
  }

  return(list(output,mapping))

}
```

```
# example usage of the unscale_unhot function
candidates_decoded <-  unscale_unhot(data_mat         = as.matrix(candidates),
                                     data_af_ohot     = srcData_prepped,
                                     Column_Spec      = Col_Spec,
                                     apply_levels     = TRUE,
                                     factorlevels     = factorlevels,
```

```
                                    apply_dateformat = TRUE,
                                    apply_isRandom   = TRUE,
                                    apply_docref     = TRUE
                                   )

candidates_decoded_data    <- candidates_decoded[[1]]
candidates_decoded_mapping <- candidates_decoded[[2]]
```

```
kable(candidates_decoded_data[1:10,])
```

	TRANS_DATE	TRANS_DATE_Mth	TRANS_DATE_DoM	TRANS_DATE_DoW	SYSTEM_DATE
649971	2014-01-17	1	17	6	2014-01-17
1312185	2018-01-25	1	25	5	2018-01-24
1874038	2014-04-30	4	30	4	2014-05-02
1738261	2018-03-16	3	16	6	2018-10-24
1658052	2012-04-25	4	25	4	2012-04-25
339225	2016-05-31	5	31	3	2016-06-03
125290	2014-10-15	10	15	4	2014-10-15
1193599	2016-02-29	2	29	2	2016-04-14
686145	2014-12-31	12	31	4	2015-01-06
1552304	2016-09-15	9	15	5	2016-09-16

```
#print(candidates_decoded_data)
```

WHY the Transaction is Unusual

Most deep learning systems are awkward to promote in real businesses because they cannot explain WHY they made a classification or decision. Hence it is difficult for colleagues to build trust in the model.

The autoencoder approach is different. We have the reconstruction error for each column, so we can order the fields in the transaction by the proportion of error they contribute to the overall reconstruction error.

Therefore, we can say why the reconstruction failed. Let's look at the above candidates to see…

```
# get squared errors for each field
candidate_row_refs <- as.numeric(candidate_row_refs)

candidate_recon_errs_byfield <- (py$Testi_py[candidate_row_refs,]
                                 -
                                 py$Testi_preds[candidate_row_refs,])^2

#We now have errors for 188 columns, but we need to map these back to the 21 params of the problem
#eg we need to ensure the 10+ onehot columns for TRANS_TYPE are represented by one error, not 10+

# create outline of mapping table, from input cols to output cols for each candidate
mapping <- cbind.data.frame(t(candidates_decoded_mapping[1,]),
                            t(candidates_decoded_mapping[2,]),
                            t(as.matrix(candidate_recon_errs_byfield)),
                            stringsAsFactors = FALSE)
# add column names
colnames(mapping) <- c("input_cols",
                       "Sequence",
                       paste0("candidate_", seq(from=1, to=nrow(candidate_recon_errs_byfield)))
```

```r
                       )

# create new column for output column names, using Sequence column of output format
mapping <- mapping %>%
            inner_join(Col_Spec %>% select(Name, Sequence), by="Sequence")

# get max error for each factor column, otherwise factored columns are over represented
# eg, trans_type has >10 columns due one hot transformation of factor.
# if we simply ask for max error by column, they may all be trans_type.
# similarly, if we simply mean() those columns, then we dilute the signal of an error in one of them
# so, max() is the best approach

max_errors  <- mapping %>%
                select(-input_cols) %>%
                  group_by(Sequence, Name) %>%
                    summarise_all(funs(max)) %>%
                      ungroup()

# create outline of max errors table
top_errors <- data.frame(Candidate      = as.numeric(),
                         Max_Error_1    = as.character(),
                         Max_Error_2    = as.character(),
                         Max_Error_3    = as.character(),
                         Sqrd_Err_Val_1 = as.character(),
                         Sqrd_Err_Val_2 = as.character(),
                         Sqrd_Err_Val_3 = as.character(),

                         stringsAsFactors=FALSE
                         )

# cycle thru the max errors, recording top 3 for each candidate
for (i in 1:length(candidate_row_refs)){

    # get reference to current candidate's max errors
    candidate_col_name <- paste0("candidate_",i)
    candidate_col_ref  <- which(colnames(max_errors) == candidate_col_name)
    candidate_col_name_desc <- paste0("desc(",candidate_col_name, ")")

    # record candidate ref
    top_errors[i,1] <- i

    # record top 3 errors (by name) for the candidate
    top_errors[i,c(2,3,4)] <- (max_errors %>%
                                 arrange_(.dots = candidate_col_name_desc) %>%
                                   select_(.dots=c("Name", candidate_col_name))
                                )[c(1,2,3),]$Name

    # record top 3 sqrd error values for the candidate
    top_errors[i,c(5,6,7)] <- as.matrix((max_errors %>%
                                          arrange_(.dots = candidate_col_name_desc) %>%
                                            select_(.dots=c("Name", candidate_col_name))
                                         )[c(1,2,3),2]
                                        )
```

```
}
```

```
# Let's see those error causes
kable(top_errors[1:10,])
```

Candidate	Max_Error_1	Max_Error_2	Max_Error_3	Sqrd_Err_Val_1	Sqrd_
1	TRANS_DATE_Mth	SYSTEM_DATE_Mth	TRANS_DATE_DoW	0.116229596950589	0.1100
2	TRANS_DATE_DoM	TRANS_DATE_Mth	SYSTEM_DATE_Mth	0.0994065575897936	0.0552
3	ACCOUNT_NO_Digit7	DOCUMENT_REF_L1	TRANS_DATE_DoM	0.838431382412439	0.3783
4	TRANS_DATE_Mth	TRANS_DATE	TRANS_DATE_DoW	0.297617317198556	0.0187
5	SYSTEM_DATE_DoM	SYSTEM_DATE	TRANS_DATE_DoW	0.11784272141687	0.0252
6	SYSTEM_DATE_DoM	TRANS_DATE_Mth	TRANS_DATE_DoW	0.288381384525243	0.1197
7	SYSTEM_DATE	TRANS_DATE_Mth	SYSTEM_DATE_Mth	0.0641468543237147	0.0522
8	TRANS_DATE_Mth	TRANS_DATE_DoW	SYSTEM_DATE_DoM	0.135347461900486	0.0876
9	TRANS_DATE_Mth	SYSTEM_DATE_Mth	SYSTEM_DATE	0.341143784204778	0.1157
10	TRANS_DATE_DoM	TRANS_DATE_Mth	SYSTEM_DATE_DoW	0.147096901389345	0.1362

```
#print(top_errors)
```

Explore the latent space representation

The bottleneck within the autoencoder contains the 'latent space' learnt by the model. As each record is compressed into the bottleneck, it is reduced to three values. These are embeddings. It is instructive to investigate these embeddings to see whether the dummy records occupy a different space to the real records.

Note, embeddings are the representation of a record in the latent space.They are not the weights of the neurons at the bottleneck, but the activations following a specific record.

```
# we compile a new model which has the same encoding architecture as the above model,
# but no decoding layers. we then apply the weights for the encoding layers from the above model
autoencoder_embeds = create_model_basic(input_dim        = input_dim,
                                        apply_dropout    = False,
                                        dropout_rate     = None,
                                        get_embeddings   = True,
                                        pretrained_model = autoencoder_basic)
```

Let's see the model for embeddings, which should be the encode section of the autoencoder, not the decode section. Remember, weights already trained and applied.

```
autoencoder_embeds.summary()
```

```
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## input_13 (InputLayer)        (None, 389)               0
## _________________________________________________________________
## dense_229 (Dense)            (None, 256)               99840
## _________________________________________________________________
## leaky_re_lu_91 (LeakyReLU)   (None, 256)               0
## _________________________________________________________________
## dense_230 (Dense)            (None, 128)               32896
## _________________________________________________________________
## leaky_re_lu_92 (LeakyReLU)   (None, 128)               0
```

```
## _________________________________________________________________
## dense_231 (Dense)            (None, 64)               8256
## _________________________________________________________________
## leaky_re_lu_93 (LeakyReLU)   (None, 64)               0
## _________________________________________________________________
## dense_232 (Dense)            (None, 32)               2080
## _________________________________________________________________
## leaky_re_lu_94 (LeakyReLU)   (None, 32)               0
## _________________________________________________________________
## dense_233 (Dense)            (None, 16)               528
## _________________________________________________________________
## leaky_re_lu_95 (LeakyReLU)   (None, 16)               0
## _________________________________________________________________
## dense_234 (Dense)            (None, 8)                136
## _________________________________________________________________
## leaky_re_lu_96 (LeakyReLU)   (None, 8)                0
## _________________________________________________________________
## dense_235 (Dense)            (None, 4)                36
## _________________________________________________________________
## leaky_re_lu_97 (LeakyReLU)   (None, 4)                0
## _________________________________________________________________
## dense_236 (Dense)            (None, 3)                15
## _________________________________________________________________
## leaky_re_lu_98 (LeakyReLU)   (None, 3)                0
## =================================================================
## Total params: 143,787
## Trainable params: 143,787
## Non-trainable params: 0
## _________________________________________________________________
```

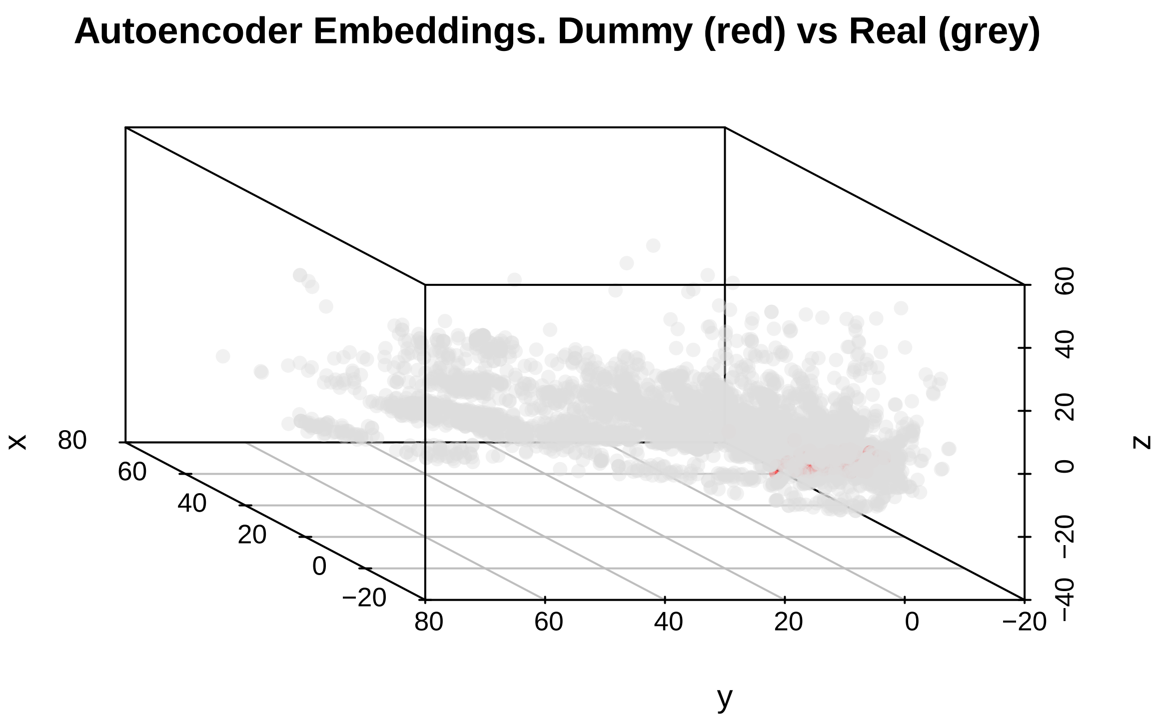
Confirmed, thats just the encode, let's compile the model and run it with the test data. We aim to get the activations at the throat of the autoencoder, which is the end of the encode section. Those activations are the 'embeddings'

```
## Compile Model with Side by Side Validation
autoencoder_embeds.compile(optimizer = 'adadelta',
                           loss      = 'binary_crossentropy',
                           # metrics   = ['accuracy']
                           )

activations = autoencoder_embeds.predict(Testi_py)
```

This outputs a numpy array of the activations, three for each record in the test set. We'll append the dummy/real flag to the records and plot in a 3D scatter plot to see if any pattern is apparent in between the two record types.

```
activations <- cbind.data.frame(isRandom = as.numeric(recon_errs$isRandom),
                                py$activations)

#There are 250k records, too many to plot
#Get sample of 5000 real records with all dummy records
activations <- rbind(activations %>% filter(isRandom == 0) %>% sample_n(5000),
                     activations %>% filter(isRandom == 1)
                    )

colnames(activations) <- c("isRandom","x","y","z")
```

```r
library(scatterplot3d)
library(scales)
colors <- c("#DDDDDD", "#FF0000")

with(activations, {
  scatterplot3d(x, y, z,
                angle = -45,
                color = alpha(colors[isRandom+1], 0.4),
                pch   = 16,
                main  = "Autoencoder Embeddings. Dummy (red) vs Real (grey)"
                #,xlim  = c(-5,25), ylim = c(-10,20), zlim = c(-10,40)
                #,xlab="", ylab="", zlab=""
                #,type="h" #drop line to xy plane for easy location.
                )
})
```



The above chart shows how the dummy records form a group, sharing similarly low y and z values, varying predominantly along the x axis. Very much a distinct group.

SQL to Export the Data From the Accounting System

For reference, below is the SQL used to extract the records from the accounting system

```sql
USE [ManagementReporting0]
GO
/****** Object:  StoredProcedure [dbo].[sp_GetAllNominalTransactions]
Script Date: 16/01/2019 12:44:38 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[sp_GetAllNominalTransactions]

AS
BEGIN
    SET NOCOUNT ON;
WITH
Customer_Refs
AS
(
    SELECT DISTINCT
        TRANS_DATE,
        TRANS_REF AS DOCUMENT_REF,
        MAX(ACCOUNT_NO_CUST) AS ACCOUNT_NO_CUST
    FROM
    (
        SELECT TRANS_DATE, TRANS_REF,
               CASE WHEN ISNUMERIC(LTRIM([ACCOUNT_NO])) = 1
                    THEN LTRIM([ACCOUNT_NO])
                    ELSE 0
                    END
                    AS ACCOUNT_NO_CUST
        FROM PUBLIC_CUSTOMER_TRANSACTIONS
        UNION
        SELECT TRANS_DATE, TRANS_REF,
               CASE WHEN ISNUMERIC(LTRIM([ACCOUNT_NO])) = 1
                    THEN LTRIM([ACCOUNT_NO])
                    ELSE 0
                    END
                    AS ACCOUNT_NO_CUST
        FROM PUBLIC_CUSTOMER_TRANSACTION_HISTORY
    ) AS DT_CUST
    GROUP BY
        TRANS_DATE,
        TRANS_REF
)
,
Supplier_Refs
AS
(
    SELECT DISTINCT
        TRANS_DATE,
        DOCUMENT_REF,
        MAX(ACCOUNT_NO_SUPP) AS ACCOUNT_NO_SUPP
    FROM
```

```
    (
        SELECT TRANS_DATE, DOCUMENT_REF,
               CASE WHEN ISNUMERIC(LTRIM([ACCOUNT_NO])) = 1
                    THEN LTRIM([ACCOUNT_NO])
                    ELSE 0
                    END
                    AS ACCOUNT_NO_SUPP
        FROM PUBLIC_SUPPLIER_TRANSACTIONS
        UNION
        SELECT TRANS_DATE, DOCUMENT_REF,
               CASE WHEN ISNUMERIC(LTRIM([ACCOUNT_NO])) = 1
                    THEN LTRIM([ACCOUNT_NO])
                    ELSE 0
                    END
                    AS ACCOUNT_NO_SUPP
        FROM PUBLIC_SUPPLIER_TRANSACTION_HISTORY
    ) AS DT_SUPP
    GROUP BY
        TRANS_DATE,
        DOCUMENT_REF
)

SELECT
    ACCOUNT_NO_Digit1,
    ACCOUNT_NO_Digit2,
    ACCOUNT_NO_Digit3,
    ACCOUNT_NO_Digit4,
    ACCOUNT_NO_Digit5,
    ACCOUNT_NO_Digit6,
    ACCOUNT_NO_Digit7,
    TRANS_DATE,
    TRANS_DATE_Mth,
    TRANS_DATE_DoM,
    TRANS_DATE_DoW,
    SYSTEM_DATE,
    SYSTEM_DATE_Mth,
    SYSTEM_DATE_DoM,
    SYSTEM_DATE_DoW,
    PostingDelayDays,
    DOCUMENT_REF_L1,
    BC_TRANS_VALUE,
    TRANS_TYPE,
    OPERATOR,
    CUST_REF,
    SUPP_REF,
    DOCUMENT_REF
FROM
(
    SELECT
        NEWID() AS RowID,
        LEFT(Curr.ACCOUNT_NO,1) AS ACCOUNT_NO_Digit1,
        SUBSTRING(Curr.ACCOUNT_NO,2,1) AS ACCOUNT_NO_Digit2,
        SUBSTRING(Curr.ACCOUNT_NO,3,1) AS ACCOUNT_NO_Digit3,
```

```
    SUBSTRING(Curr.ACCOUNT_NO,4,1) AS ACCOUNT_NO_Digit4,
    SUBSTRING(Curr.ACCOUNT_NO,5,1) AS ACCOUNT_NO_Digit5,
    SUBSTRING(Curr.ACCOUNT_NO,6,1) AS ACCOUNT_NO_Digit6,
    RIGHT(Curr.ACCOUNT_NO,1) AS ACCOUNT_NO_Digit7,
    Curr.TRANS_DATE,
    DATEPART(mm,Curr.TRANS_DATE) AS TRANS_DATE_Mth,
    DATEPART(dd,Curr.TRANS_DATE) AS TRANS_DATE_DoM,
    DATEPART(dw,Curr.TRANS_DATE) AS TRANS_DATE_DoW,
    Curr.SYSTEM_DATE,
    DATEPART(mm,Curr.SYSTEM_DATE) AS SYSTEM_DATE_Mth,
    DATEPART(dd,Curr.SYSTEM_DATE) AS SYSTEM_DATE_DoM,
    DATEPART(dw,Curr.SYSTEM_DATE) AS SYSTEM_DATE_DoW,
    DATEDIFF(dd,Curr.TRANS_DATE, Curr.SYSTEM_DATE) AS PostingDelayDays,
    LEFT(Curr.DOCUMENT_REF,1) AS DOCUMENT_REF_L1,
    BC_TRANS_VALUE,
    Curr.TRANS_TYPE,
    CASE WHEN OPERATOR = '    ' THEN 'XX' ELSE OPERATOR END AS OPERATOR,
    Curr.DOCUMENT_REF,
    COALESCE(Customer_Refs.ACCOUNT_NO_CUST,-1) AS CUST_REF,
    COALESCE(Supplier_Refs.ACCOUNT_NO_SUPP,-1) AS SUPP_REF
FROM
    [PUBLIC_NOMINAL_TRANSACTIONS] AS Curr
    LEFT OUTER JOIN
    [17_NominalTransactionCurrentMajorKey] AS Key_Curr
    ON Curr.ACCOUNT_NO = Key_Curr.ACCOUNT_NO
    AND Curr.TRANS_DATE = Key_Curr.TRANS_DATE
    AND Curr.DOCUMENT_REF = Key_Curr.DOCUMENT_REF

    LEFT OUTER JOIN
    Supplier_Refs
    ON Curr.TRANS_DATE = Supplier_Refs.TRANS_DATE
    AND Curr.DOCUMENT_REF = Supplier_Refs.DOCUMENT_REF

    LEFT OUTER JOIN
    Customer_Refs
    ON Curr.TRANS_DATE = Customer_Refs.TRANS_DATE
    AND Curr.DOCUMENT_REF = Customer_Refs.DOCUMENT_REF

UNION ALL

SELECT
    NEWID() AS RowID,
    LEFT(Hist.ACCOUNT_NO,1) AS ACCOUNT_NO_Digit1,
    SUBSTRING(Hist.ACCOUNT_NO,2,1) AS ACCOUNT_NO_Digit2,
    SUBSTRING(Hist.ACCOUNT_NO,3,1) AS ACCOUNT_NO_Digit3,
    SUBSTRING(Hist.ACCOUNT_NO,4,1) AS ACCOUNT_NO_Digit4,
    SUBSTRING(Hist.ACCOUNT_NO,5,1) AS ACCOUNT_NO_Digit5,
    SUBSTRING(Hist.ACCOUNT_NO,6,1) AS ACCOUNT_NO_Digit6,
    RIGHT(Hist.ACCOUNT_NO,1) AS ACCOUNT_NO_Digit7,
    Hist.TRANS_DATE,
    DATEPART(mm,Hist.TRANS_DATE) AS TRANS_DATE_Mth,
    DATEPART(dd,Hist.TRANS_DATE) AS TRANS_DATE_DoM,
    DATEPART(dw,Hist.TRANS_DATE) AS TRANS_DATE_DoW,
```

```
        Hist.SYSTEM_DATE,
        DATEPART(mm,Hist.SYSTEM_DATE) AS SYSTEM_DATE_Mth,
        DATEPART(dd,Hist.SYSTEM_DATE) AS SYSTEM_DATE_DoM,
        DATEPART(dw,Hist.SYSTEM_DATE) AS SYSTEM_DATE_DoW,
        DATEDIFF(dd,Hist.TRANS_DATE, Hist.SYSTEM_DATE) AS PostingDelayDays,
        LEFT(Hist.DOCUMENT_REF,1) AS DOCUMENT_REF_L1,
        BC_TRANS_VALUE,
        TRANS_TYPE,
        CASE WHEN OPERATOR = '    ' THEN 'XX' ELSE OPERATOR END AS OPERATOR,
        Hist.DOCUMENT_REF,
        COALESCE(Customer_Refs.ACCOUNT_NO_CUST,-1) AS CUST_REF,
        COALESCE(Supplier_Refs.ACCOUNT_NO_SUPP,-1) AS SUPP_REF

    FROM
        [PUBLIC_NOMINAL_TRANSACTION_HISTORY] AS Hist

        LEFT OUTER JOIN
        [16_NominalTransactionHistoryMajorKey] AS Key_Hist
        ON Hist.ACCOUNT_NO = Key_Hist.ACCOUNT_NO
        AND Hist.TRANS_DATE = Key_Hist.TRANS_DATE
        AND Hist.DOCUMENT_REF = Key_Hist.DOCUMENT_REF

        LEFT OUTER JOIN
        Supplier_Refs
        ON Hist.TRANS_DATE = Supplier_Refs.TRANS_DATE
        AND Hist.DOCUMENT_REF = Supplier_Refs.DOCUMENT_REF

        LEFT OUTER JOIN
        Customer_Refs
        ON Hist.TRANS_DATE = Customer_Refs.TRANS_DATE
        AND Hist.DOCUMENT_REF = Customer_Refs.DOCUMENT_REF
) AS DT1

ORDER BY RowID

END
```