# Deep Learning Classification with Tensorflow and Keras

Loren Collingwood

Deep learning is one of the most recent developments in machine learning based off tenserflow python package ported via the keras R package. In essence, we are building off the neural networks approach to machine learning. Overall, this process is most similar to the supervised machine learning algorithms/approach found in RTextTools but is more updated. We will work through an example from the tensorflow/rstudio website. I'll be honest, some of you may not be able to line your computer up to the right specifications in order to get tensorflow and keras to work.

## Step 1

Install and download packages, then load example data.

```
options(scipen = 999, digits = 4)

############
# Packages #
############

#install.packages(c("keras", "purrr", "pins", "tensorflow"))

library(keras)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(purrr)
library(pins)
library(tensorflow)

# Had to create a kaggle account and then download this api key #
# Otherwise you can download the data yourself and store it some place nice
board_register_kaggle(token = "~/Downloads/kaggle.json")

# Pull in the data and movie review .csv file
path <- pins::pin_get("nltkdata/movie-review", "kaggle")[1]
```

```r
# Read in the .csv file from the path
df <- readr::read_csv(path)
```

```
## Parsed with column specification:
## cols(
##   fold_id = col_double(),
##   cv_tag = col_character(),
##   html_id = col_double(),
##   sent_id = col_double(),
##   text = col_character(),
##   tag = col_character()
## )
```

```r
# take a look at first 6 rows
head(df)
```

```
## # A tibble: 6 x 6
##   fold_id cv_tag html_id sent_id text                                          tag
##     <dbl> <chr>    <dbl>   <dbl> <chr>                                         <chr>
## 1       0 cv000    29590       0 "films adapted from comic books have had~ pos
## 2       0 cv000    29590       1 "for starters , it was created by alan m~ pos
## 3       0 cv000    29590       2 "to say moore and campbell thoroughly re~ pos
## 4       0 cv000    29590       3 "the book ( or \" graphic novel , \" if ~ pos
## 5       0 cv000    29590       4 "in other words , don't dismiss this fil~ pos
## 6       0 cv000    29590       5 "if you can get past the whole comic boo~ pos
```

```r
# Look at text 1 for shites and giggles
df$text[1]
```

```
## [1] "films adapted from comic books have had plenty of success , whether they're about superheroes (
```

```r
# Count of review variable #
df %>% count(tag)
```

```
## # A tibble: 2 x 2
##   tag       n
##   <chr> <int>
## 1 neg   31783
## 2 pos   32937
```

```r
# Take a look at distribution of Words in Each Review
df$text %>%
    strsplit(" ") %>%
    sapply(length) %>%
    summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    14.0    21.0    23.1    30.0   179.0
```

## Step 2

Create training and testing datasets. We will use these to evaluate model accuracy.

```r
# Create Unique id, 80% of dataset; used for separating train/test set
training_id <- sample.int(nrow(df), size = nrow(df)*0.8)
```

```r
# Generate Train and Test sets
training <- df[training_id,]
testing <- df[-training_id,]
```

## Step 3

Define Text Vectorization Level. This will install miniconda; then install tensorflow(). You might have some problem with installation here depending on your computer setup and your computer knowledge.

```r
# Create Text Vectorization Layer #
num_words <- 10000 # you can mix it up here but we'll start with this
max_length <- 50

text_vectorization <- layer_text_vectorization(
    max_tokens = num_words,
    output_sequence_length = max_length,
)

text_vectorization %>%
    adapt(df$text)

# Look at vocabulary/words #
# TODO see https://github.com/tensorflow/tensorflow/pull/34529
head ( get_vocabulary(text_vectorization) )
```

```
## [1] "the" "a"   "and" "of"  "to"  "is"
```

```r
#You can see how the text vectorization layer transforms it's inputs:
text_vectorization(matrix(df$text[1], ncol = 1))
```

```
## tf.Tensor(
## [[  68 2835   30  359 1662   33   91 1056    5  632  631  321   41 7803
##    709 4865 1767   48 7600 1337  398 5161   48    2    1 1808 1800  148
##     17  140  109   90   69    3  359  408   40   30  503  142    0    0
##      0    0    0    0    0    0    0    0]], shape=(1, 50), dtype=int64)
```

## Step 4

Build models inputs and outputs then send to keras_model() function.

```r
# Build Model for Input
input <- layer_input(shape = 1, dtype = "string")

# Output
output <- input %>%
    text_vectorization() %>%
    layer_embedding(input_dim = num_words + 1, output_dim = 16) %>%
    layer_global_average_pooling_1d() %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dropout(0.5) %>%
    layer_dense(units = 1, activation = "sigmoid")
```

```
model <- keras_model(input, output)

# Hidden Units and Loss Optimizer #
model %>% compile(
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics = list('accuracy')
)
```

## Step 5

Train the model real good using the fit() function.

```
###################
# Train the Model #
###################

history <- model %>% fit(
    training$text,
    as.numeric(training$tag == "pos"),
    epochs = 10,
    batch_size = 512,
    validation_split = 0.2,
    verbose=2
)
```

## Step 6

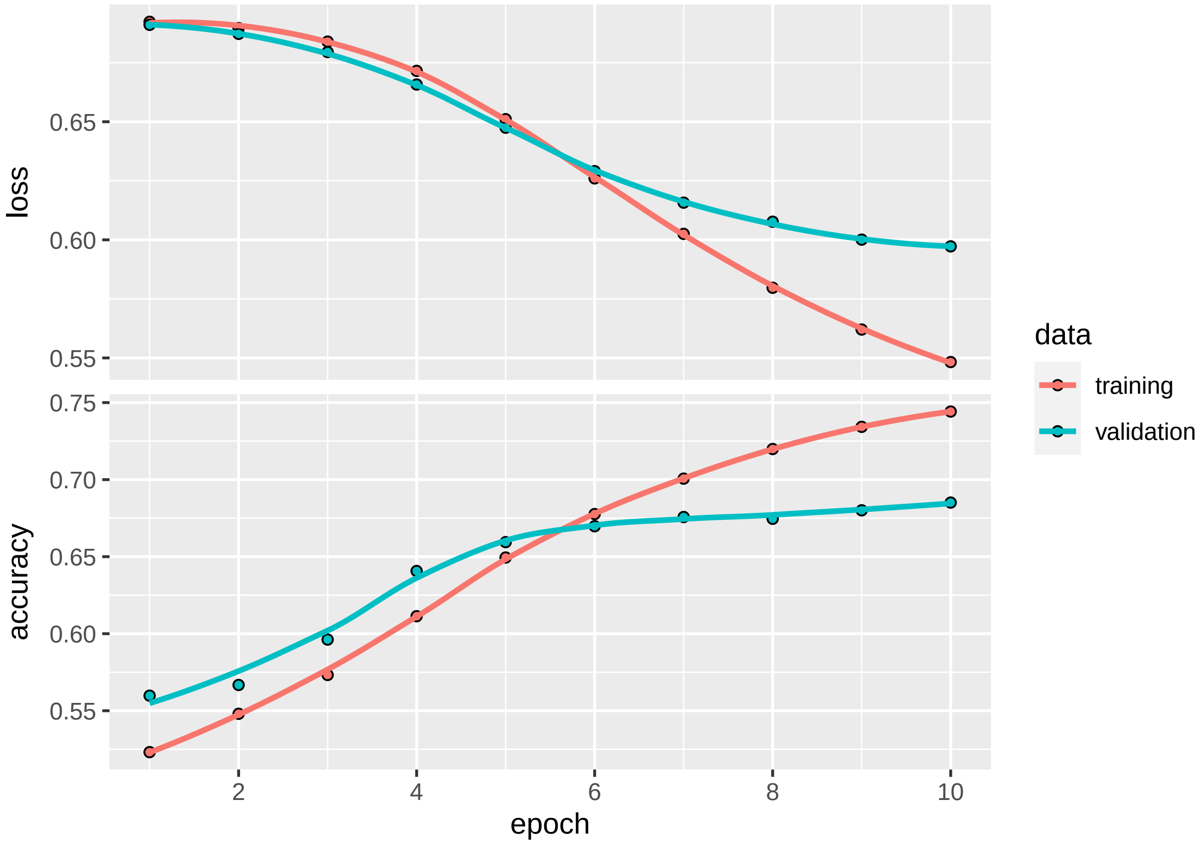
Then look at the results and extract predictions on virgin texts as needed.

```
results <- model %>% evaluate(testing$text, as.numeric(testing$tag == "pos"), verbose = 0)
results
```

```
##     loss accuracy
##   0.6015   0.6780
```

```
# Look at the within model accuracy/loss
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
###################
# Predict Classes #
###################

# This is what you would do onto virgin text

head(pred_out <- as.data.frame(predict(model,
                  x = testing$text)))
```

```
##       V1
## 1 0.2534
## 2 0.4921
## 3 0.3944
## 4 0.4229
## 5 0.4727
## 6 0.2452
```

```
# Create the dummy/binary where 1 = pos, 0 = negative (rating)
pred_out$pred_bin <- ifelse(pred_out$V1 >=0.5, 1, 0)

# Just add on the original truth #
pred_out$truth <- as.numeric(testing$tag == "pos")

# Look at Confusion Matrix #
table(pred_out$pred_bin, pred_out$truth)
```

```
##
##        0    1
##    0 4457 2259
##    1 1909 4319
```

```
# Run the numbers real nice #
(4371 + 4404) / nrow(testing)
```

```
## [1] 0.6779
```