Using word embeddings

Abstract

In this Assignment, we are going to explore word embeddings by using embedding layer and pretrained embedding layer, and observe the variations in the Validation Accuracy. We look at imdb movie reviews dataset, one withembedding layer, which is already available in keras library and one with pre-trained word embeddings which is precomputed by using different machine learning task. At the end we plot graph by showing variation in validation accuracy by changing taining sample in the model.

There are two ways to obtain word embeddings:

- Learn word embeddings jointly with the main task you care about (e.g. document classification or sentiment prediction). In this setup, you would start with random word vectors, then learn your word vectors in the same way that you learn the weights of a neural network.
- Load into your model word embeddings that were pre-computed using a different machine learning task than the one you are trying to solve. These are called "pre-trained word embeddings".

Let's take a look at both.

Learning word embeddings with an embedding layer

Hide

```r
library(keras)

# The embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer <- layer_embedding(input_dim = 1000, output_dim = 64)
```

A `layer_embedding()` is best understood as a dictionary that maps integer indices (which stand for specific words) to dense vectors. It takes integers as input, it looks up these integers in an internal dictionary, and it returns the associated vectors.

Let's apply this idea to the IMDB movie-review sentiment-prediction task that you're already familiar with. First, you'll quickly prepare the data. You'll restrict the movie reviews to the top 10,000 most common words (as you did the first time you worked with this dataset) and cut off the reviews after only 150 words. The network will learn 8-dimensional embeddings for each of the 10,000 words, turn the input integer sequences (2D integer tensor) into embedded sequences (3D float tensor), flatten the tensor to 2D, and train a single dense layer on top for classification.

```r
# Number of words to consider as features
max_features <- 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen <- 150

# Load the data as lists of integers.
imdb <- dataset_imdb(num_words = max_features)
c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train <- pad_sequences(x_train, maxlen = maxlen)
x_test <- pad_sequences(x_test, maxlen = maxlen)
```

Using pre-trained word embeddings

There are various pre-computed databases of word embeddings that can download and start using in a Keras embedding layer. Word2Vec is one of them. Another popular one is called "GloVe", developed by Stanford researchers in 2014. Let's take a look at how you can get started using GloVe embeddings in a Keras model.

Putting it all together: from raw text to word embeddings

We will be using a model similar to the one we just went over – embedding sentences in sequences of vectors, flattening them and training a dense layer on top. But we will do it using pre-trained word embeddings, and instead of using the pre-tokenized IMDB data packaged in Keras, we will start from scratch, by downloading the original text data.

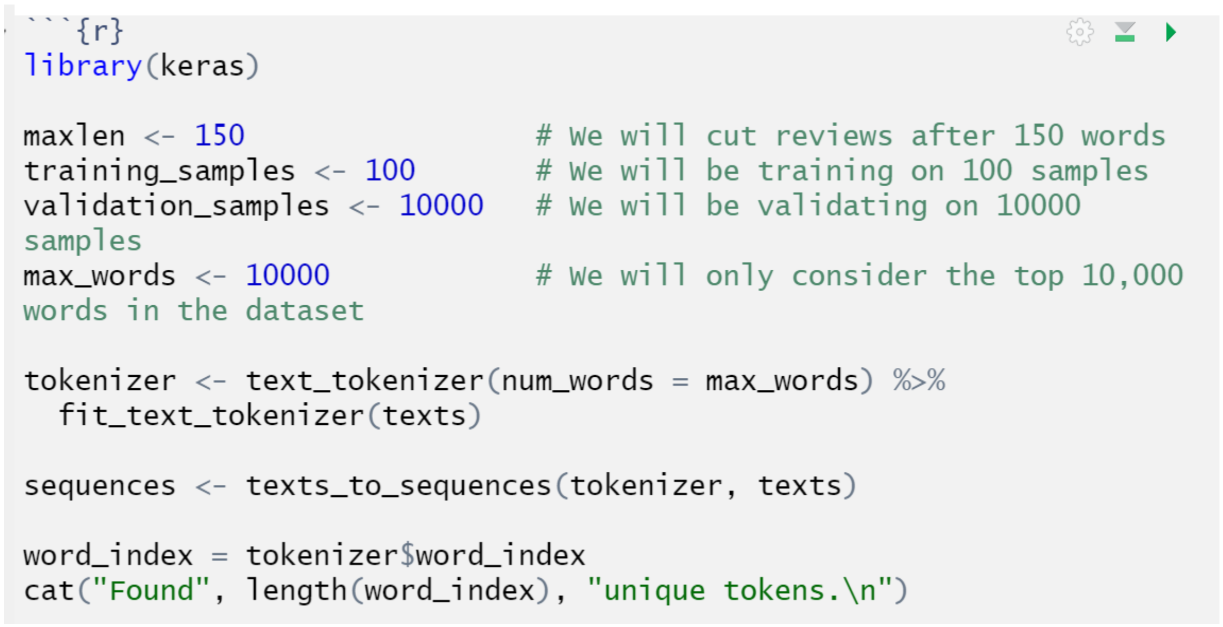
Download the IMDB data as raw text

```r
imdb_dir <- "C:/Users/Sushmita Singh/Desktop/Ad. ML/aclImdb"
train_dir <- file.path(imdb_dir, "train")

labels <- c()
texts <- c()

for (label_type in c("neg", "pos")) {
  label <- switch(label_type, neg = 0, pos = 1)
  dir_name <- file.path(train_dir, label_type)
  for (fname in list.files(dir_name, pattern = glob2rx("*.txt"),
                           full.names = TRUE)) {
    texts <- c(texts, readChar(fname, file.info(fname)$size))
    labels <- c(labels, label)
  }
}
```

Tokenize the data

Let's vectorize the texts we collected, and prepare a training and validation split.

Because pre-trained word embeddings are meant to be particularly useful on problems where little training data is available.



~~~
```{r}
library(keras)

maxlen <- 150                 # We will cut reviews after 150 words
training_samples <- 100       # We will be training on 100 samples
validation_samples <- 10000   # We will be validating on 10000 
samples
max_words <- 10000            # We will only consider the top 10,000 
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")
~~~

```
Found 88584 unique tokens.
```

Hide

```
data <- pad_sequences(sequences, maxlen = maxlen)

labels <- as.array(labels)
cat("Shape of data tensor:", dim(data), "\n")
```

```
Shape of data tensor: 25000 150
```

Hide

```
cat('Shape of label tensor:', dim(labels), "\n")
```

```
Shape of label tensor: 25000
```

Hide

```
# Split the data into a training set and a validation set
# But first, shuffle the data, since we started from data
# where sample are ordered (all negative first, then all positive).
indices <- sample(1:nrow(data))
training_indices <- indices[1:training_samples]
validation_indices <- indices[(training_samples + 1):
                              (training_samples + validation_samples)]

x_train <- data[training_indices,]
y_train <- labels[training_indices]
```

```
x_val <- data[validation_indices,]
y_val <- labels[validation_indices]
```

Download the GloVe word embeddings

Pre-process the embeddings

Let's parse the un-zipped file (it's a `txt` file) to build an index mapping words (as strings) to their vector representation (as number vectors).

Hide

```
glove_dir = 'C:/Users/Sushmita Singh/Desktop/Ad. ML/glove.6B'
lines <- readLines(file.path(glove_dir, "glove.6B.100d.txt"))

embeddings_index <- new.env(hash = TRUE, parent = emptyenv())
for (i in 1:length(lines)) {
  line <- lines[[i]]
  values <- strsplit(line, " ")[[1]]
  word <- values[[1]]
  embeddings_index[[word]] <- as.double(values[-1])
}

cat("Found", length(embeddings_index), "word vectors.\n")
```

```
Found 400000 word vectors.
```

Next, you'll build an embedding matrix that you can load into an embedding layer. It must be a matrix of shape `(max_words, embedding_dim)`, where each entry *i* contains the `embedding_dim`-dimensional vector for the word of index *i* in the reference word index (built during tokenization). Note that index 1 isn't supposed to stand for any word or token – it's a placeholder.

Next, you'll build an embedding matrix that you can load into an embedding layer. It must be a matrix of shape `(max_words, embedding_dim)`, where each entry *i* contains the `embedding_dim`-dimensional vector for the word of index *i* in the reference word index (built during tokenization). Note that index 1 isn't supposed to stand for any word or token – it's a placeholder.

Hide

```
embedding_dim <- 100

embedding_matrix <- array(0, c(max_words, embedding_dim))

for (word in names(word_index)) {
  index <- word_index[[word]]
  if (index < max_words) {
    embedding_vector <- embeddings_index[[word]]
    if (!is.null(embedding_vector))
      # Words not found in the embedding index will be all zeros.
      embedding_matrix[index+1,] <- embedding_vector
  }
}
```

Define a model

We will be using the same model architecture as before:

Hide

Hide

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

summary(model)
```

```
Model: "sequential_1"
___________________________________________________________
Layer (type)             Output Shape             Param #
===========================================================
embedding_2 (Embedding)  (None, 150, 100)         1000000
___________________________________________________________
flatten_1 (Flatten)      (None, 15000)            0
___________________________________________________________
dense_1 (Dense)          (None, 32)               480032
___________________________________________________________
dense_2 (Dense)          (None, 1)                33
===========================================================
Total params: 1,480,065
Trainable params: 1,480,065
Non-trainable params: 0
___________________________________________________________
```

Load the GloVe embeddings in the model

Load the GloVe matrix you prepared into the embedding layer, the first layer in the model.

```
get_layer(model, index = 1) %>%
  set_weights(list(embedding_matrix)) %>%
  freeze_weights()
```

Additionally, you'll freeze the weights of the embedding layer.

Train and evaluate

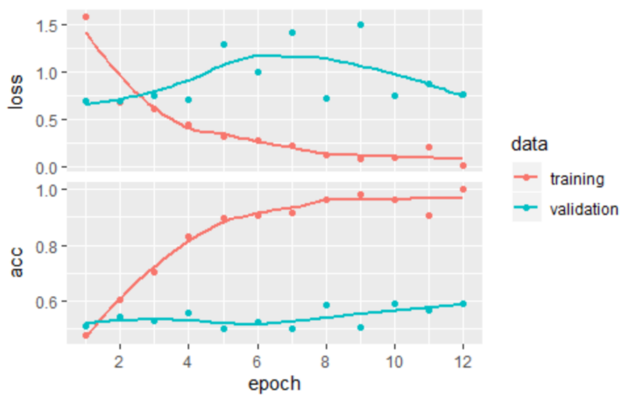
Let's compile our model and train it:

Hide

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history <- model %>% fit(
  x_train, y_train,
  epochs = 12,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
save_model_weights_hdf5(model, "pre_trained_glove_model.h5")
```

Let's plot its performance over time:

```
plot(history)
```



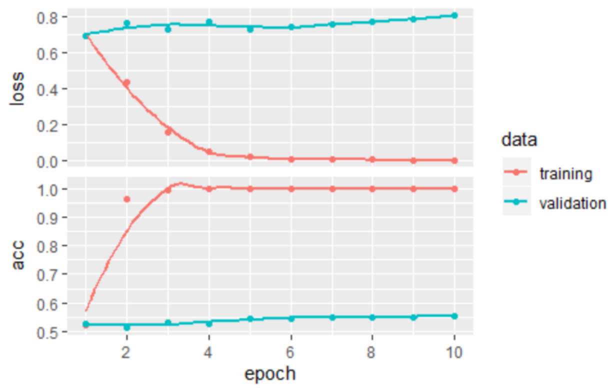
The model quickly starts overfitting, unsurprisingly given the small number of training samples. Validation accuracy has high variance for the same reason, but seems to reach high 50s.

Hide

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history <- model %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
```

```
plot(history)
```



Validation accuracy stalls in the low 50s.

Finally, let's evaluate the model on the test data. First, we will need to tokenize the test data:

Hide

```
test_dir <- file.path(imdb_dir, "test")

labels <- c()
texts <- c()

for (label_type in c("neg", "pos")) {
  label <- switch(label_type, neg = 0, pos = 1)
  dir_name <- file.path(test_dir, label_type)
  for (fname in list.files(dir_name, pattern = glob2rx("*.txt"),
                           full.names = TRUE)) {
    texts <- c(texts, readChar(fname, file.info(fname)$size))
    labels <- c(labels, label)
  }
}

sequences <- texts_to_sequences(tokenizer, texts)
x_test <- pad_sequences(sequences, maxlen = maxlen)
y_test <- as.array(labels)
```

And let's load and evaluate the first model:

Hide

```
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```

```
$loss
[1] 0.9239654

$acc
[1] 0.55692
```

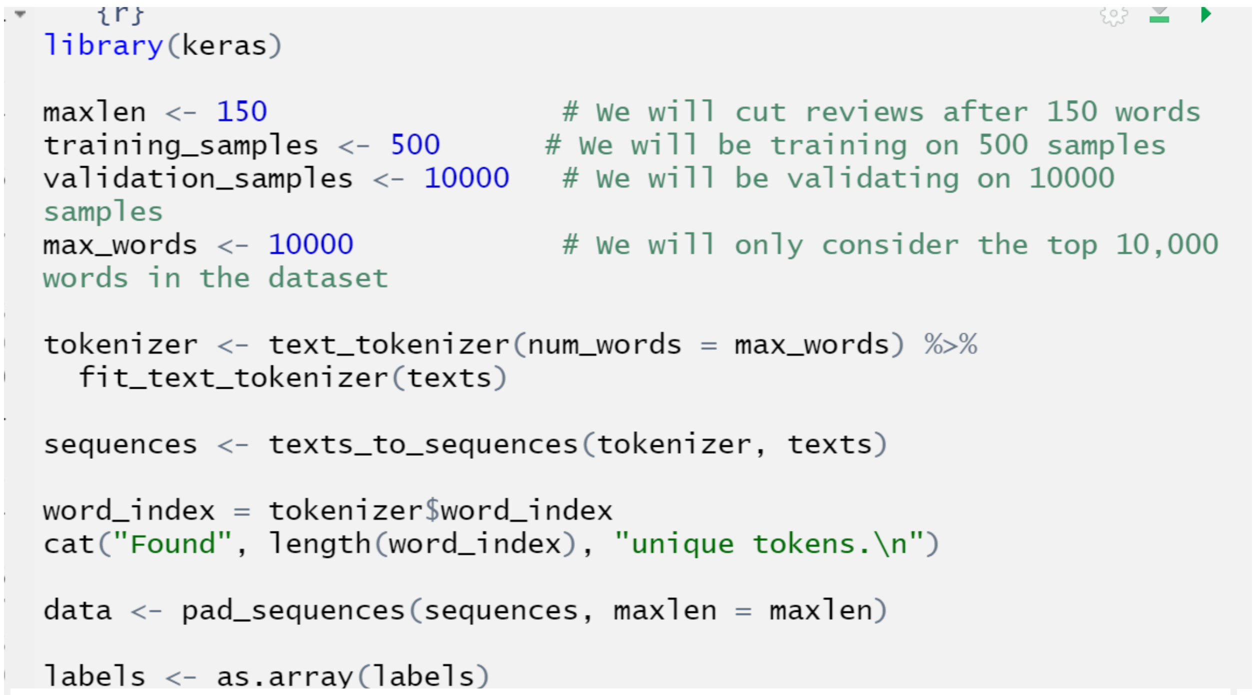
We get an appalling test accuracy of 55.6%.

Consider both a embedding layer, and a pretrained word embedding.

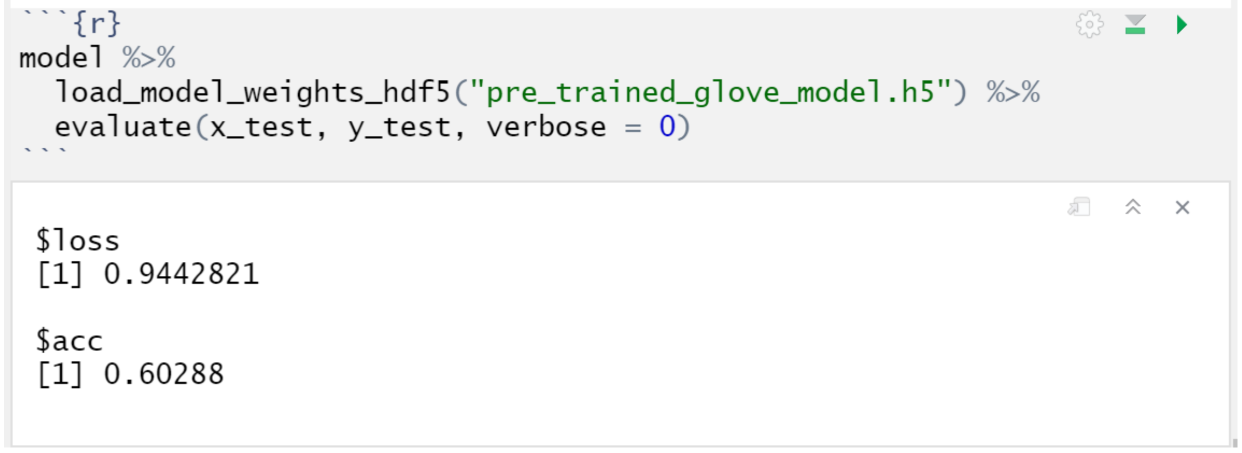
With embedding layer We get validation accuracy of ~87% With pretrained layer Validation accuracy stalls in the 50s. So embedding layes gives better performance.

Now let’s try changing the number of training samples to determine at what point the embedding layer gives better performance.

1) **Training Sample = 500**

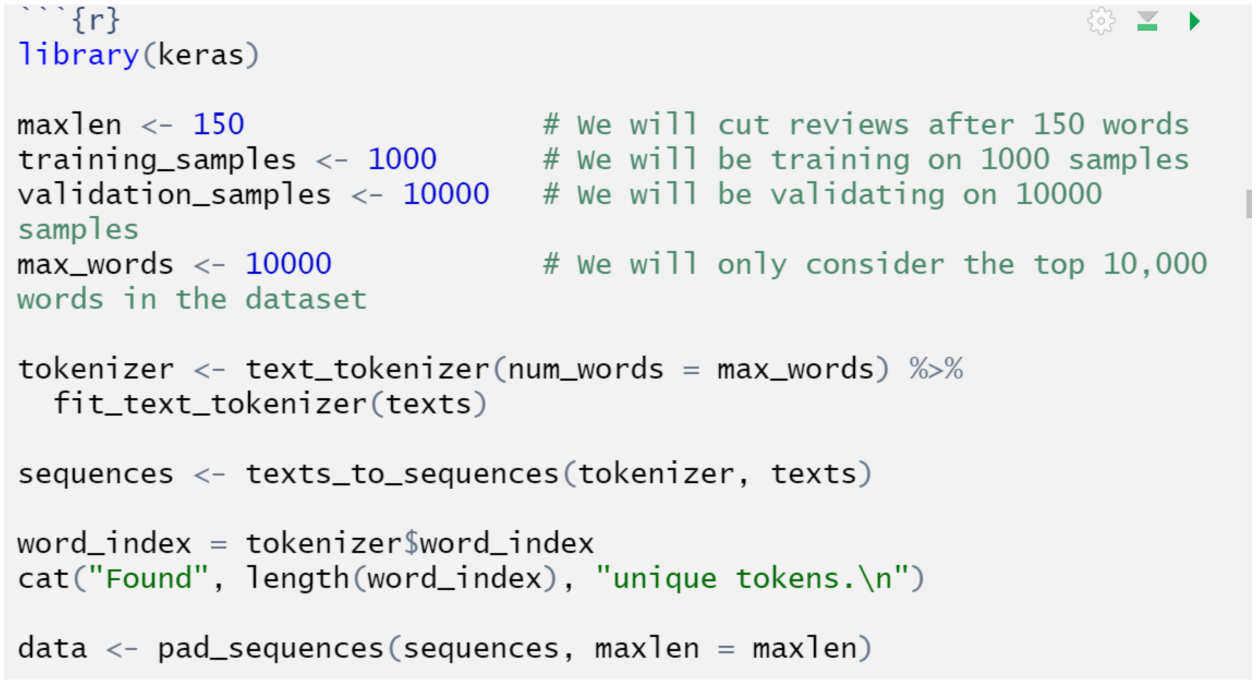


```
  {r}
library(keras)

maxlen <- 150                   # We will cut reviews after 150 words
training_samples <- 500        # We will be training on 500 samples
validation_samples <- 10000     # We will be validating on 10000
samples
max_words <- 10000              # We will only consider the top 10,000
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)

labels <- as.array(labels)
```

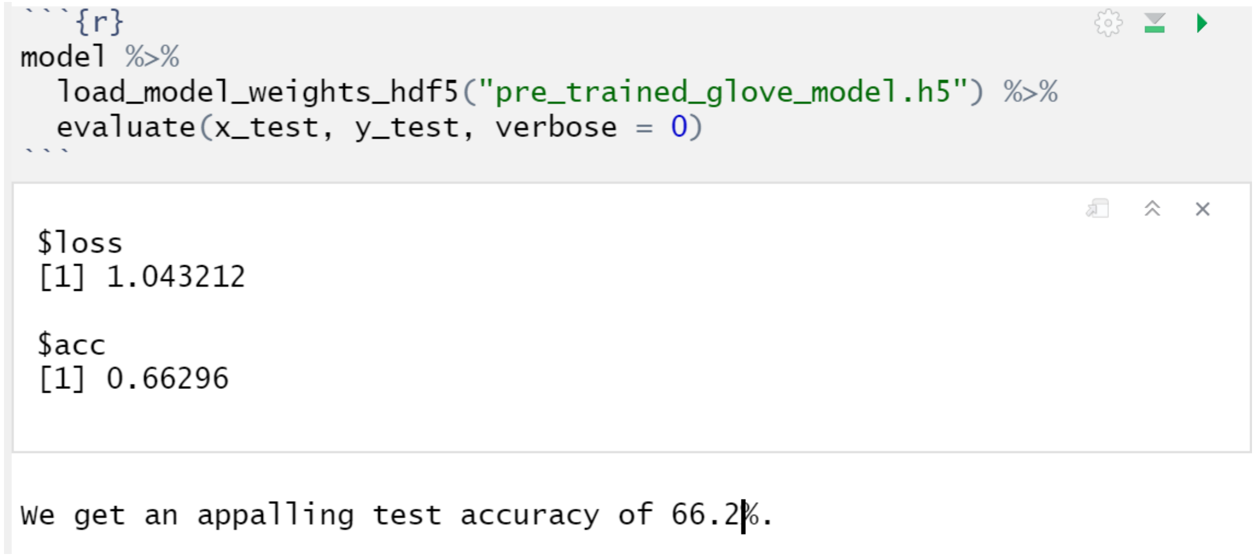


~~~
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```

$loss
[1] 0.9442821

$acc
[1] 0.60288
~~~

2) Training Sample = 1000



~~~
```{r}
library(keras)

maxlen <- 150                 # We will cut reviews after 150 words
training_samples <- 1000      # We will be training on 1000 samples
validation_samples <- 10000   # We will be validating on 10000 
samples
max_words <- 10000            # We will only consider the top 10,000 
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)
~~~

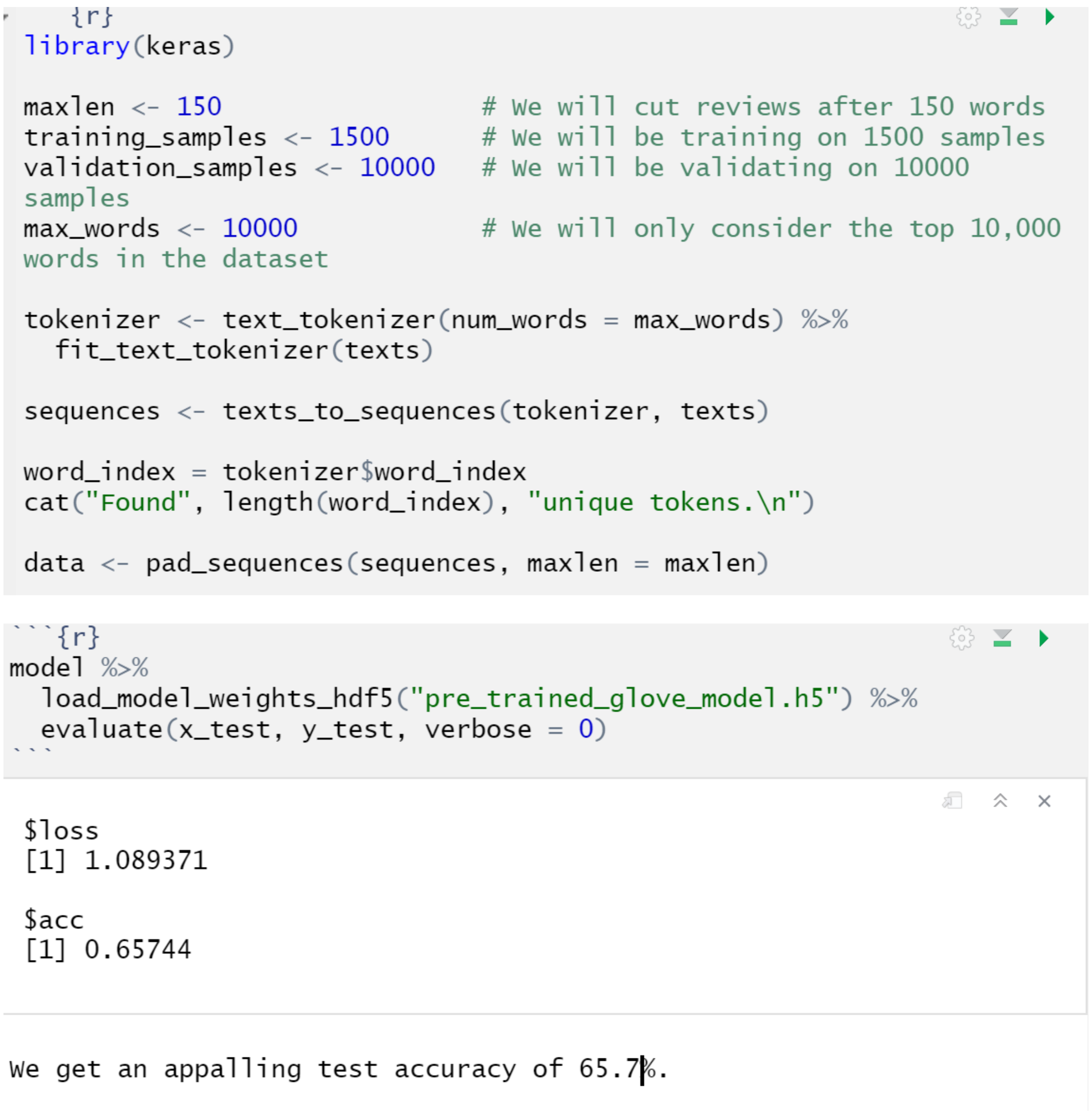


~~~
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```
~~~

```
$loss
[1] 1.043212

$acc
[1] 0.66296
```

We get an appalling test accuracy of 66.2%.

3) Training Sample = 1500



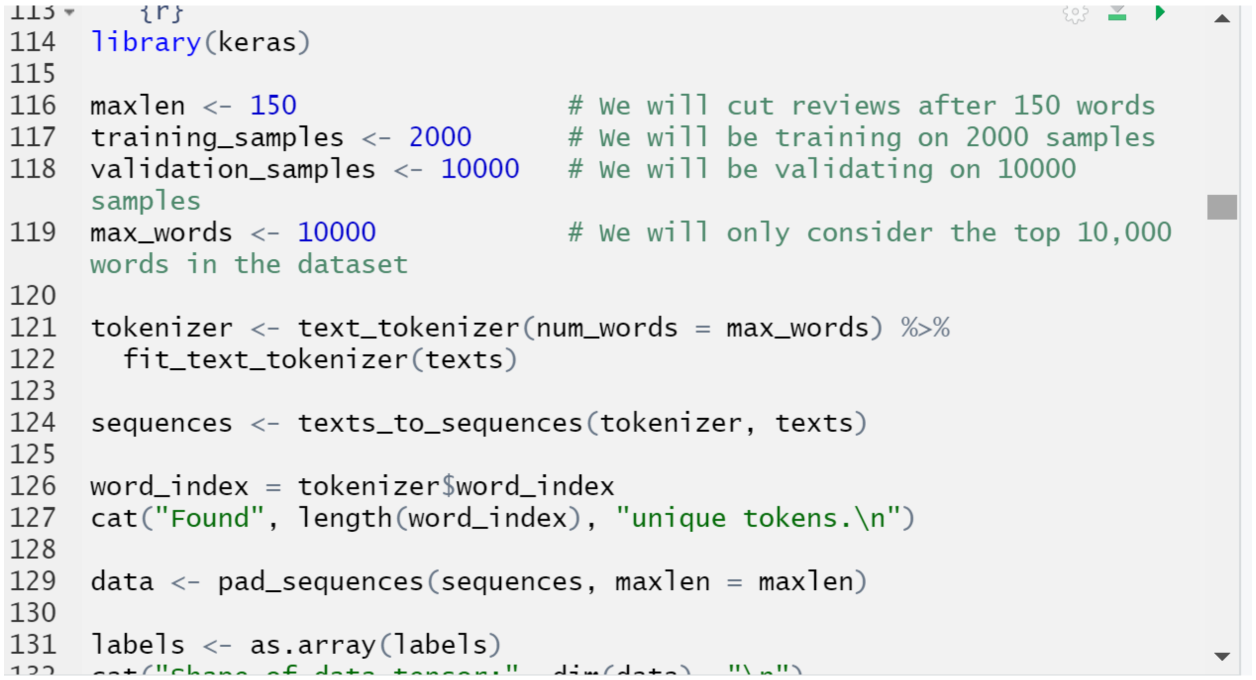
```
   {r}
library(keras)

maxlen <- 150                 # We will cut reviews after 150 words
training_samples <- 1500      # We will be training on 1500 samples
validation_samples <- 10000   # We will be validating on 10000
samples
max_words <- 10000            # We will only consider the top 10,000
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)
```

~~~
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```
~~~

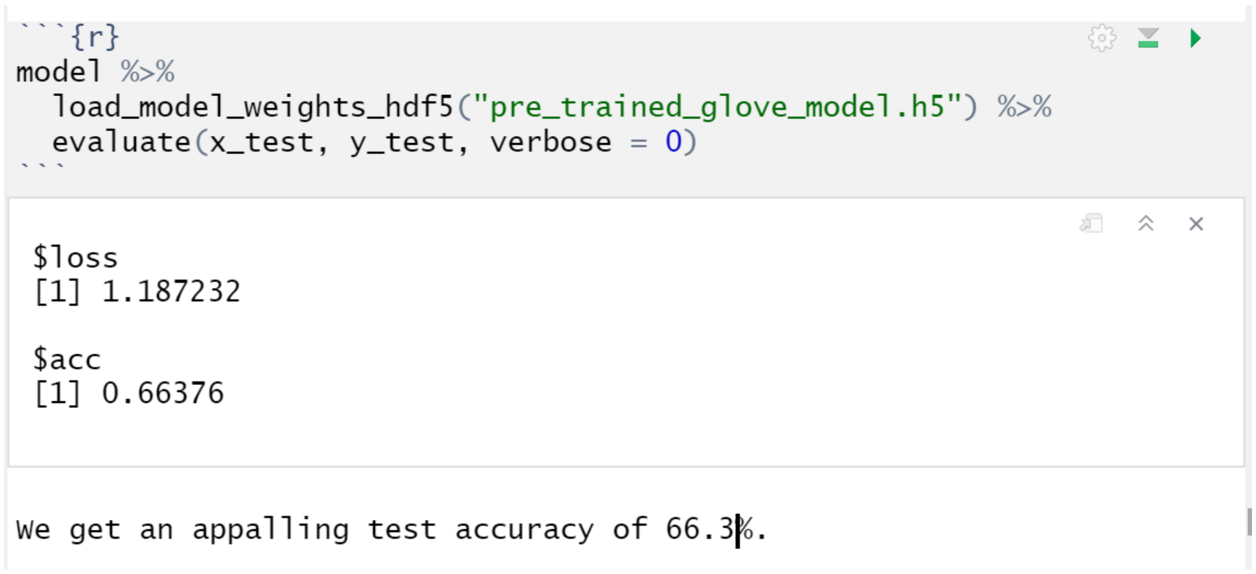
```
$loss
[1] 1.089371

$acc
[1] 0.65744
```

We get an appalling test accuracy of 65.7%.

4) Training Sample = 2000



~~~
```{r}
library(keras)

maxlen <- 150                   # We will cut reviews after 150 words
training_samples <- 2000        # We will be training on 2000 samples
validation_samples <- 10000     # We will be validating on 10000
samples
max_words <- 10000              # We will only consider the top 10,000
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)

labels <- as.array(labels)
cat("Shape of data tensor:", dim(data), "\n")
~~~

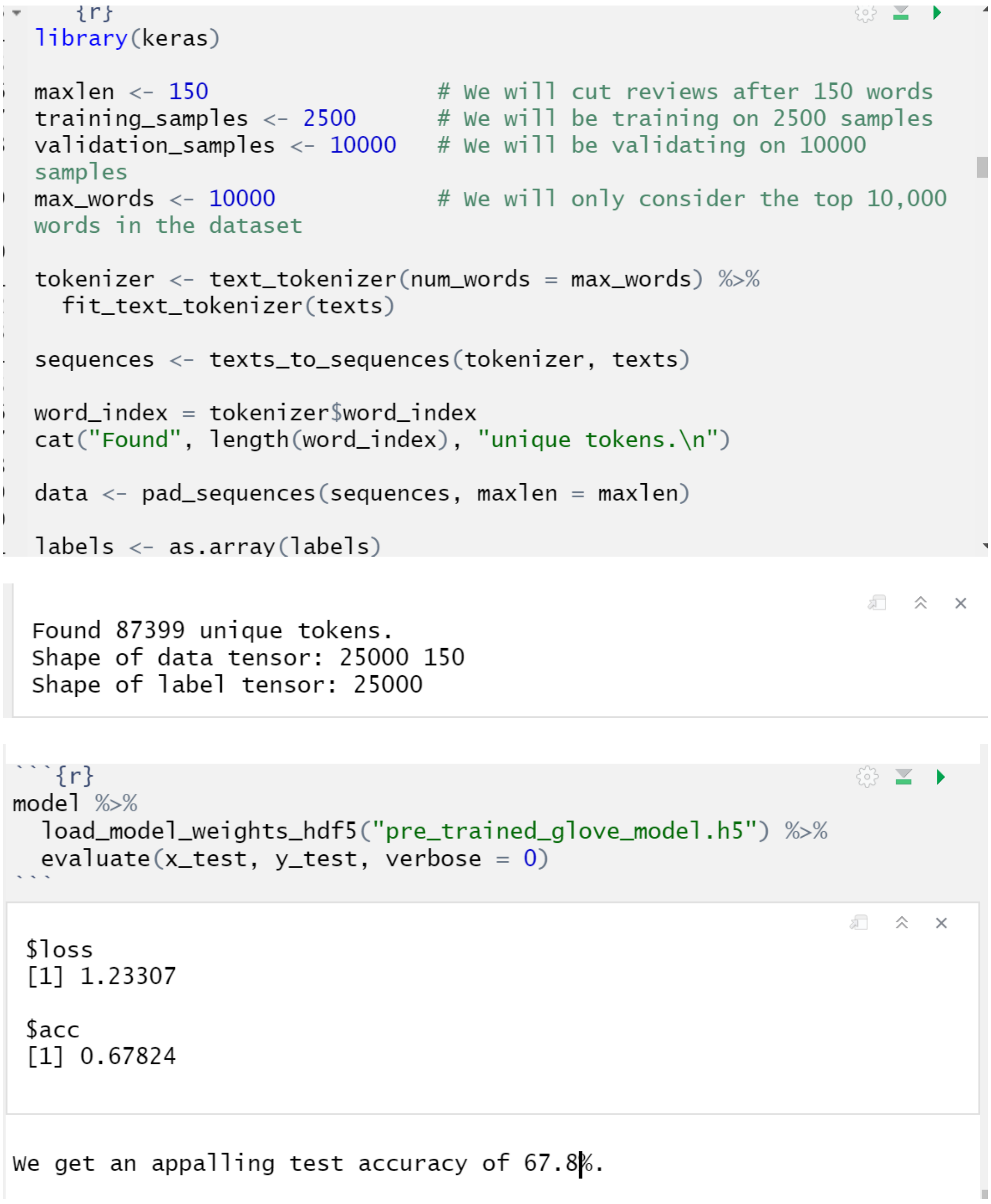


~~~
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```
~~~

```
$loss
[1] 1.187232

$acc
[1] 0.66376
```

We get an appalling test accuracy of 66.3%.

5) Training Sample = 2500



```
  {r}
library(keras)

maxlen <- 150                    # We will cut reviews after 150 words
training_samples <- 2500         # We will be training on 2500 samples
validation_samples <- 10000      # We will be validating on 10000 
samples
max_words <- 10000               # We will only consider the top 10,000 
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)

labels <- as.array(labels)
```

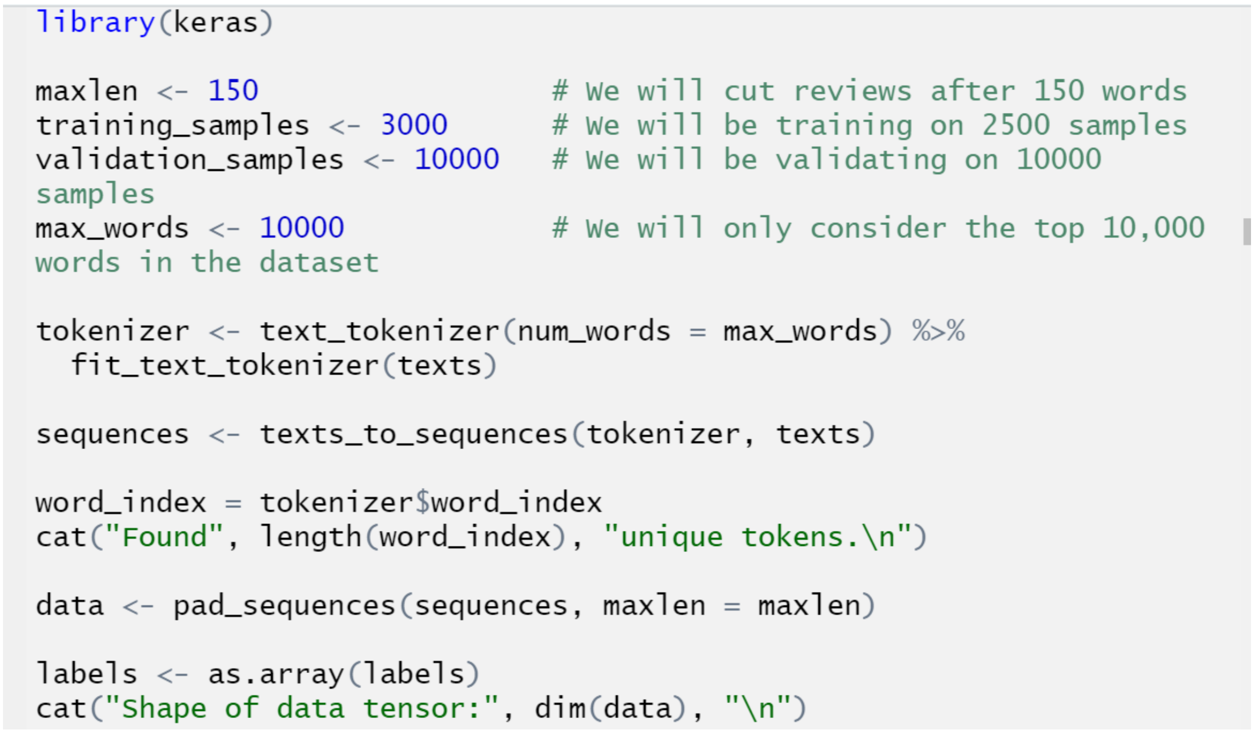
```
Found 87399 unique tokens.
Shape of data tensor: 25000 150
Shape of label tensor: 25000
```

~~~
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```
~~~

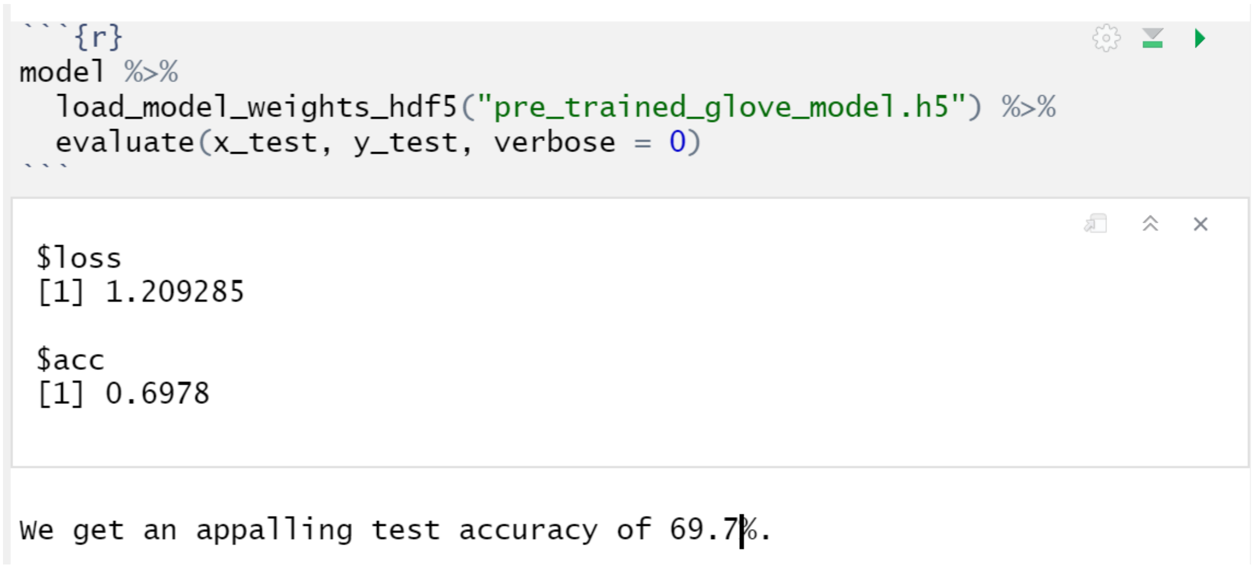
```
$loss
[1] 1.23307

$acc
[1] 0.67824
```

We get an appalling test accuracy of 67.8%.

6) Training Sample = 3000



```r
library(keras)

maxlen <- 150                  # We will cut reviews after 150 words
training_samples <- 3000       # We will be training on 2500 samples
validation_samples <- 10000    # We will be validating on 10000
samples
max_words <- 10000             # We will only consider the top 10,000
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)

labels <- as.array(labels)
cat("Shape of data tensor:", dim(data), "\n")
```

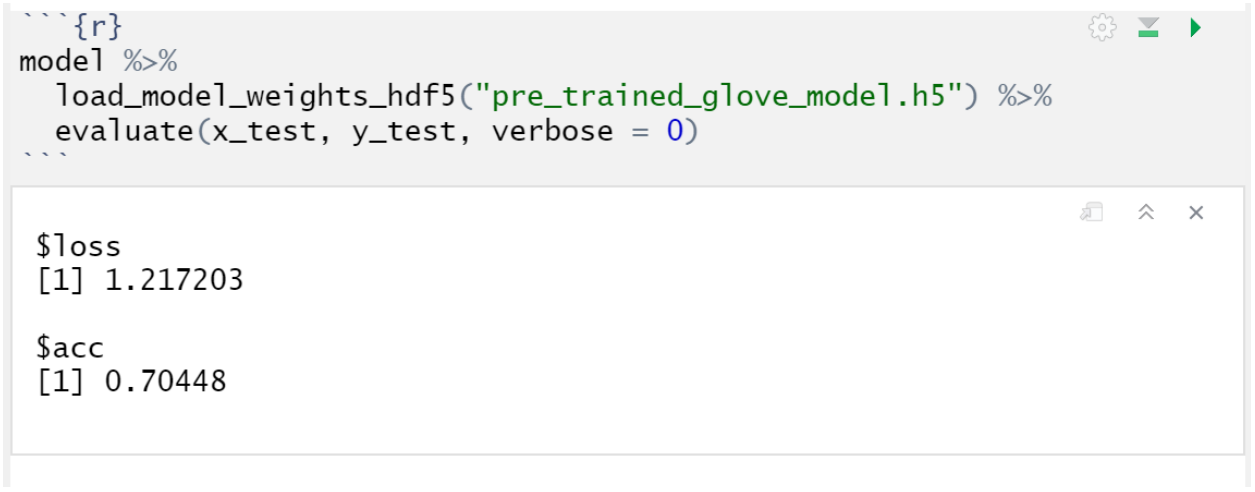


~~~r
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
~~~
```

```
$loss
[1] 1.209285

$acc
[1] 0.6978
```

We get an appalling test accuracy of 69.7%.
```

7) Training Sample = 3500

```
library(keras)

maxlen <- 150                 # We will cut reviews after 150 words
training_samples <- 3500      # We will be training on 3500 samples
validation_samples <- 10000   # We will be validating on 10000 
samples
max_words <- 10000            # We will only consider the top 10,000 
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)

labels <- as.array(labels)
cat("Shape of data tensor:", dim(data), "\n")
```



~~~
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```
~~~

```
$loss
[1] 1.217203

$acc
[1] 0.70448
```

8) Training Sample = 4500

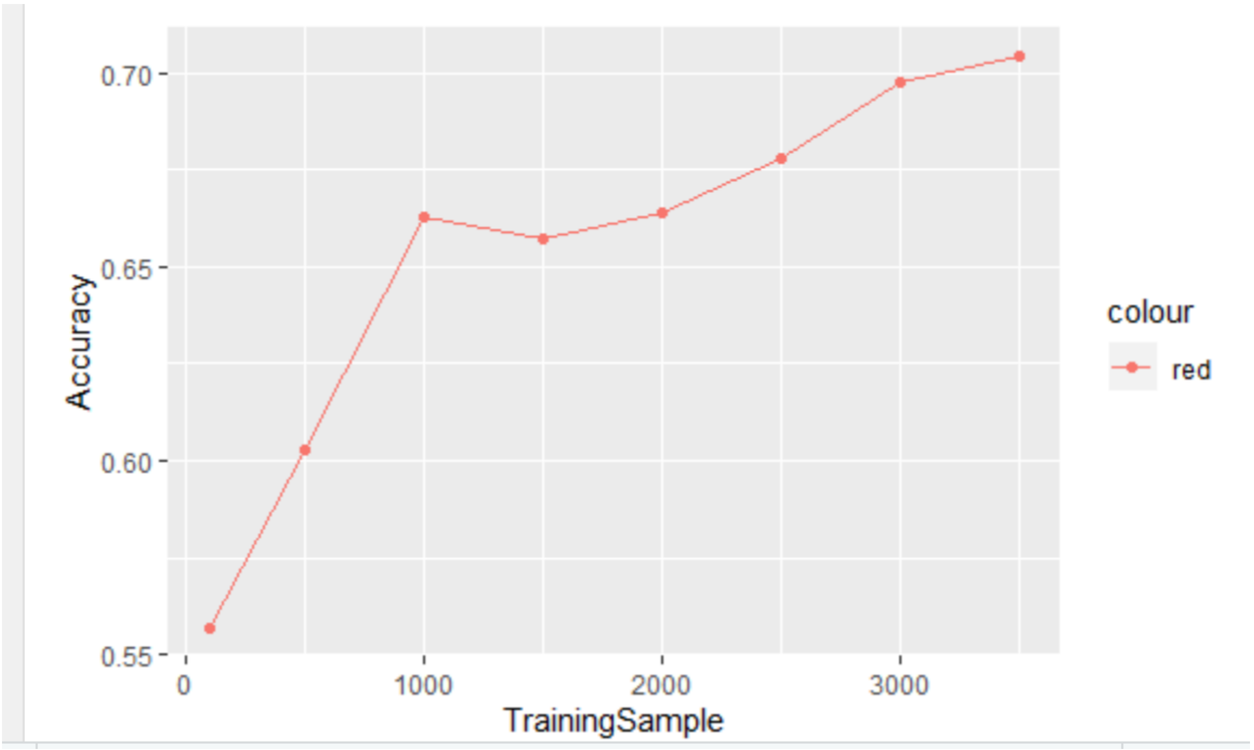
~~~
```{r}
library(keras)

maxlen <- 150                 # We will cut reviews after 150 words
training_samples <- 4500      # We will be training on 4500 samples
validation_samples <- 10000   # We will be validating on 10000
samples
max_words <- 10000            # We will only consider the top 10,000
words in the dataset

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)

word_index = tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")

data <- pad_sequences(sequences, maxlen = maxlen)
~~~

~~~
```{r}
model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test, verbose = 0)
```
~~~

```
$loss
[1] 1.284046

$acc
[1] 0.7342
```

We get an appalling test accuracy of 73.4%.

~~~
```{r}
library(ggplot2)
ggplot(data =Graph, aes(x = TrainingSample, y = Accuracy, col =
"red")) +
  geom_line() +
  geom_point()
```
~~~



0.70
0.65
0.60
0.55
Accuracy
0
1000
2000
3000
TrainingSample
colour
red