Assignment 2 - RNN on IMDB Dataset

Author: Steven Spence

Assignment 2 – Recurrent Neural Network (RNN) - IMDB

Consider the IMDB example from Chapter 6. Re-run the example modifying the following:

Cutoff reviews after 150 words Restrict training samples to 100 Validate on 10,000 samples Consider only the top 10,000 words

Consider both a embedding layer, and a pretrained word embedding. Which approach did better? Now try changing the number of training samples to determine at what point the embedding layer gives better performance.

Initially, the pre-trained word embedding model (GloVe in this case) was outperformed by the single embedding layer model. However, as the summary chart below shows, the single embedded layer model surpassed the pre-trained model when the training sample size exceeded 750.

![A screenshot of a map

Description automatically generated]()

See code below that was utilized to gather the performance results for each level of training samples.

IMDB dataset was downloaded from: “<http://mng.bz/0tIo>” The GloVe pre-trained model was downloaded from: “<https://nlp.stanford.edu/projects/glove>”

Download and tokenize the IMDB dataset with this chunk of code. “Training\_Sample” was altered from this chunk of code to increase the value and observe the results.

# Download the IMDB dataset from "http://mng.bz/0tIo"  
  
require(keras)  
  
imdb\_dir <- "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)  
 }  
}  
  
maxlen <- 150 # Cutoff reviews after 150 words  
training\_samples <- 100 # Train on 100 samples  
validation\_samples <- 10000 # Validates on 10,000 samples  
max\_words <- 10000 # Consider only 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")  
cat("Shape of label tensor:", dim(labels), "\n")  
  
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 and integrate the GloVe embedding information via this chunk. There were a total of 400,000 word vectors.

# Download GloVe Embeddings from "https://nlp.stanford.edu/projects/glove"  
  
glove\_dir <- "glove"  
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")

Pre-process and integrate the word embeddings from GloVe via this chunk of code.

# Preprocess the embeddings  
  
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))  
 embedding\_matrix[index + 1, ] <- embedding\_vector  
 }  
}

Chunk of code utilized for the pre-trained model utilizing GloVe and the single-layer embedding model.

# Defining a pre-trained model  
  
model\_pretrained <- 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\_pretrained)  
  
# Defining a model with embedding layer that has not been pre-trained  
  
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)

This chunk of code was used to integrate the weights from GloVe and freeze the weights for this pre-trained model.

# load pre-trained embedding layer  
  
get\_layer(model\_pretrained, index = 1) %>%  
 set\_weights(list(embedding\_matrix)) %>%  
 freeze\_weights()

Train the model and verify the validation accuracy and loss for the pretrained model.

set.seed(322020)  
  
# Training and evaluation of pre-trained model  
  
model\_pretrained %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history <- model\_pretrained %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val, y\_val)  
)

Train the model and verify the validation accuracy and loss for the single embedding layer model.

set.seed(322020)  
  
# Training and evaluation of model without pre-training  
  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history <- model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val, y\_val)  
)

Summary of results are shown in the table at the beginning of the file.