Assignment 3

Ethan Luster

11/21/2021

```
# The following code is the setup for python in RStudio
library(reticulate)
use_condaenv('r-reticulate')
library(tensorflow)
library(keras)
tf$constant("Hellow")
```

```
## tf.Tensor(b'Hellow', shape=(), dtype=string)
```

```
maxlen <- 150 # maximum length: 150 words
max_features <- 10000 # maximum features: 10,000
training_samples <- 100 # maximum training samples: 100
validation_samples <- 10000 #maximum validation samples: 10000
max_words <- 10000 # maximum words used: 10000
```

The raw data comes from "https://mng.bz/0tlo" which was provided in the textbook.This data set had no folders named "train", only folders named "train". As this was the only data I could find, I used the folders in aclImdb

```
imdb_dir <- "C:/Users/13303/Documents/aclImdb/aclImdb/test" # No "training" folder in aclImdb
train_dir <- file.path(imdb_dir)
labels <- c()
texts <- c()
# sets up appropriate labels
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)
  }
}

# Tokenizes the words
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 72637 unique tokens.
```

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

```
## Shape of data tensor: 17243 150
```

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

```
## Shape of label tensor: 17243
```

```
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]
```

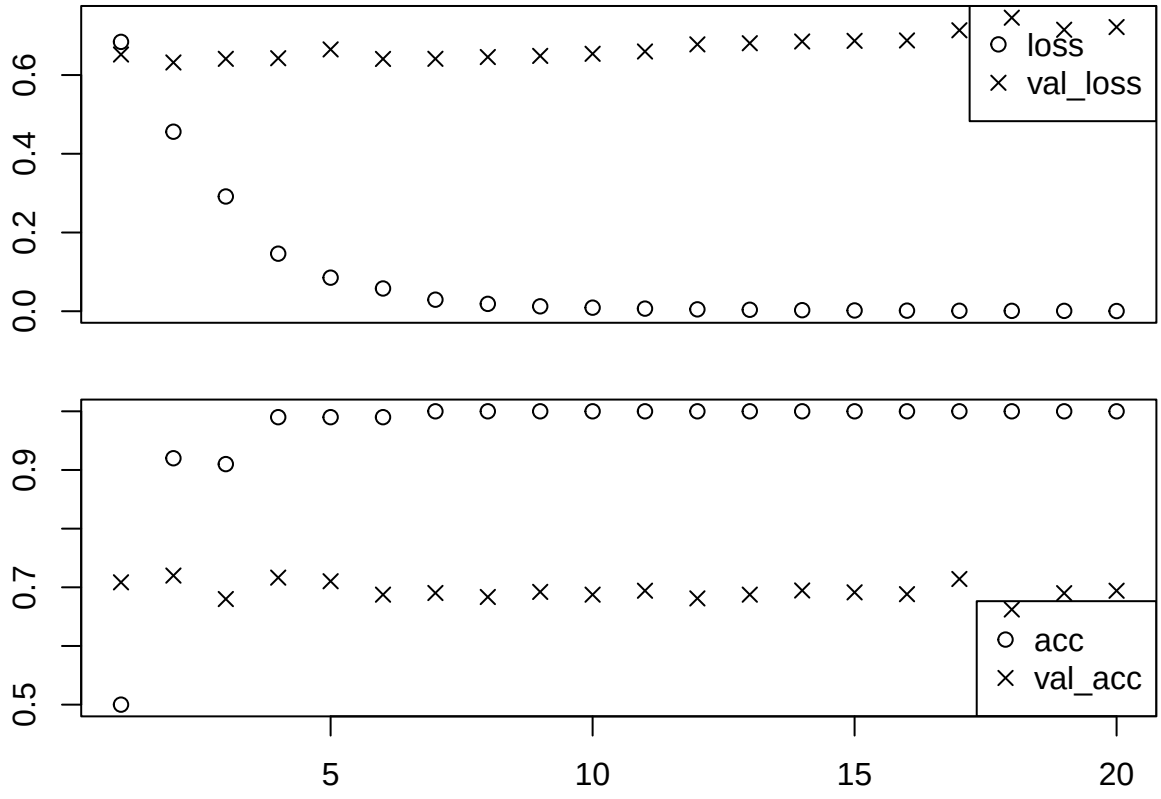
```
# Downloads and sets up glove, which creates the embedding index for the embedding layers
glove_dir <- "C:/Users/13303/Downloads/glove_dir"
lines <- readLines(file.path(glove_dir, "glove.6B.100d.txt"))
embeddings_index <- new.env(hash = TRUE, parent = emptyenv())
for (i in 1:length(lines)) {
   line <- 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.
```

```
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
  }
}
```

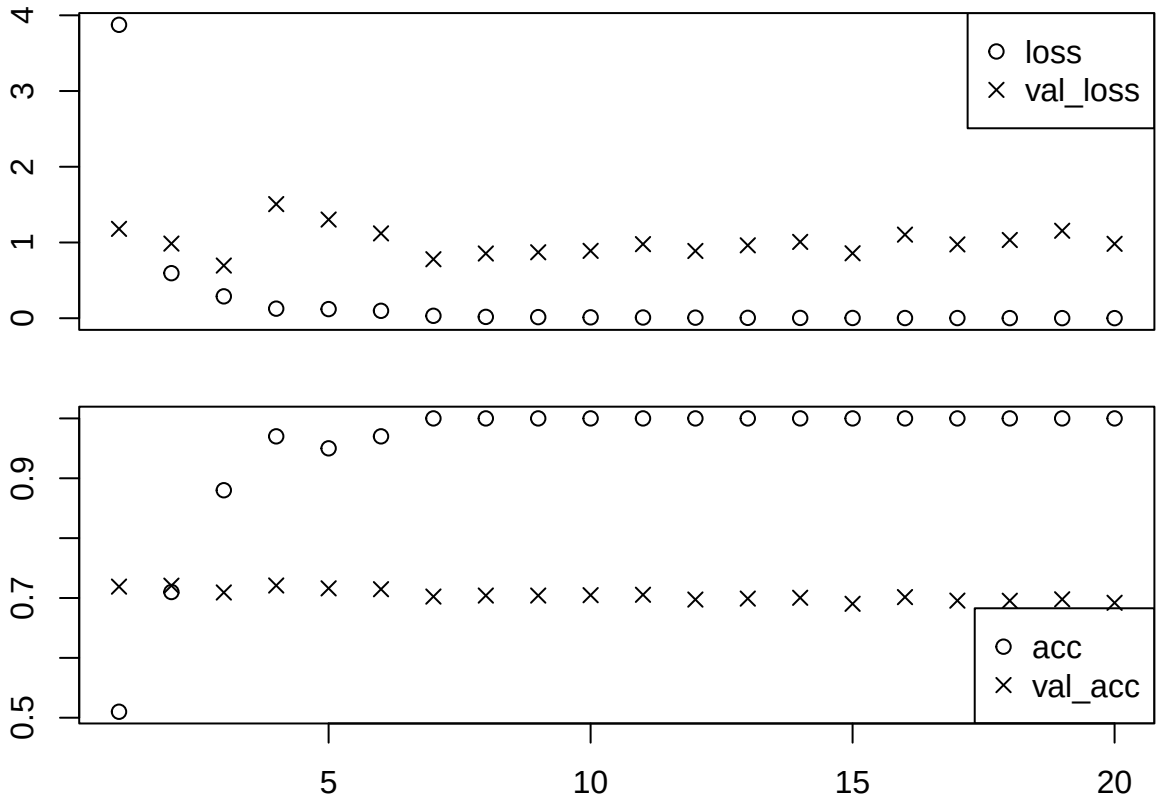
```
# First embedding layer model with 100 training samples
model_emb <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim, input_length = maxlen) %>%
  layer_flatten() %>%
```

```
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model_emb %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
history <- model_emb %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history)
```




```
# First pretrained layer model with 100 training samples
model_pre <- 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")

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

```
  freeze_weights()

model_pre %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
history <- model_pre %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history)
```



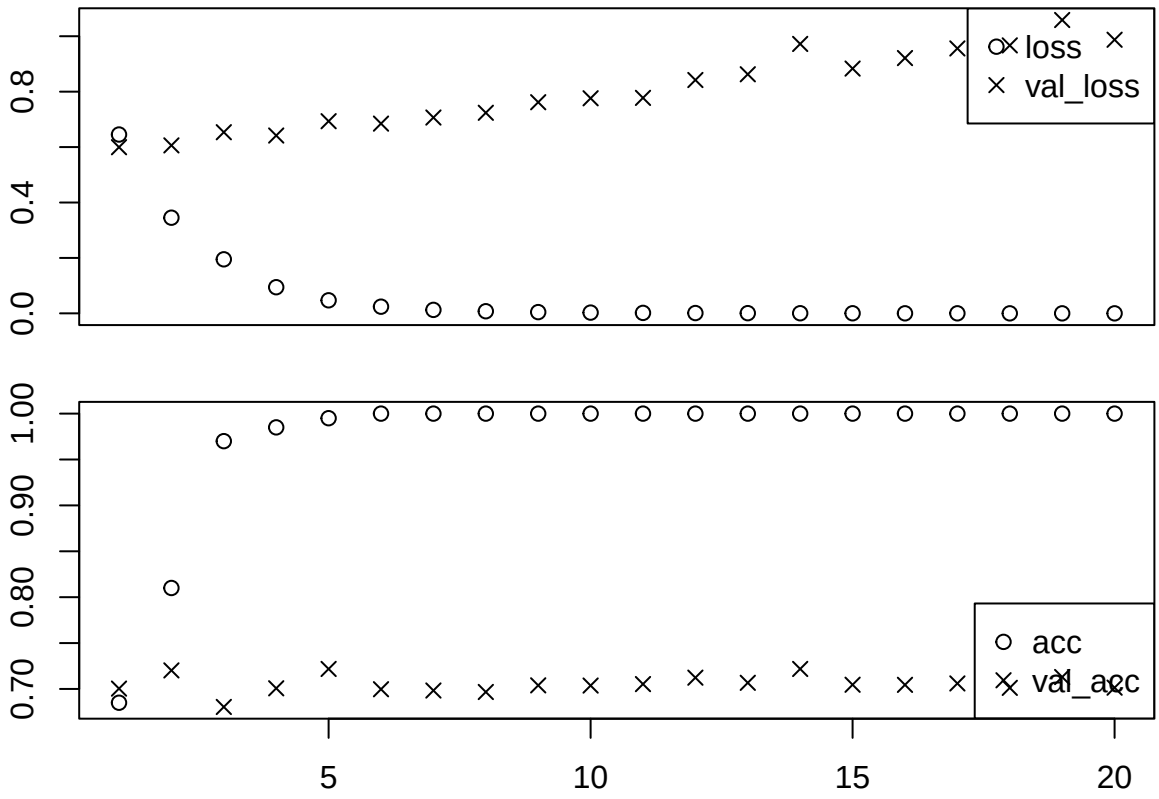
The embedding model has a validation accuarcy of .7217. The pretrained model has a validation accuarcy of .7205.

The same models will be run again, except with 200 training samples

```
training_samples <- 200
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,]
```

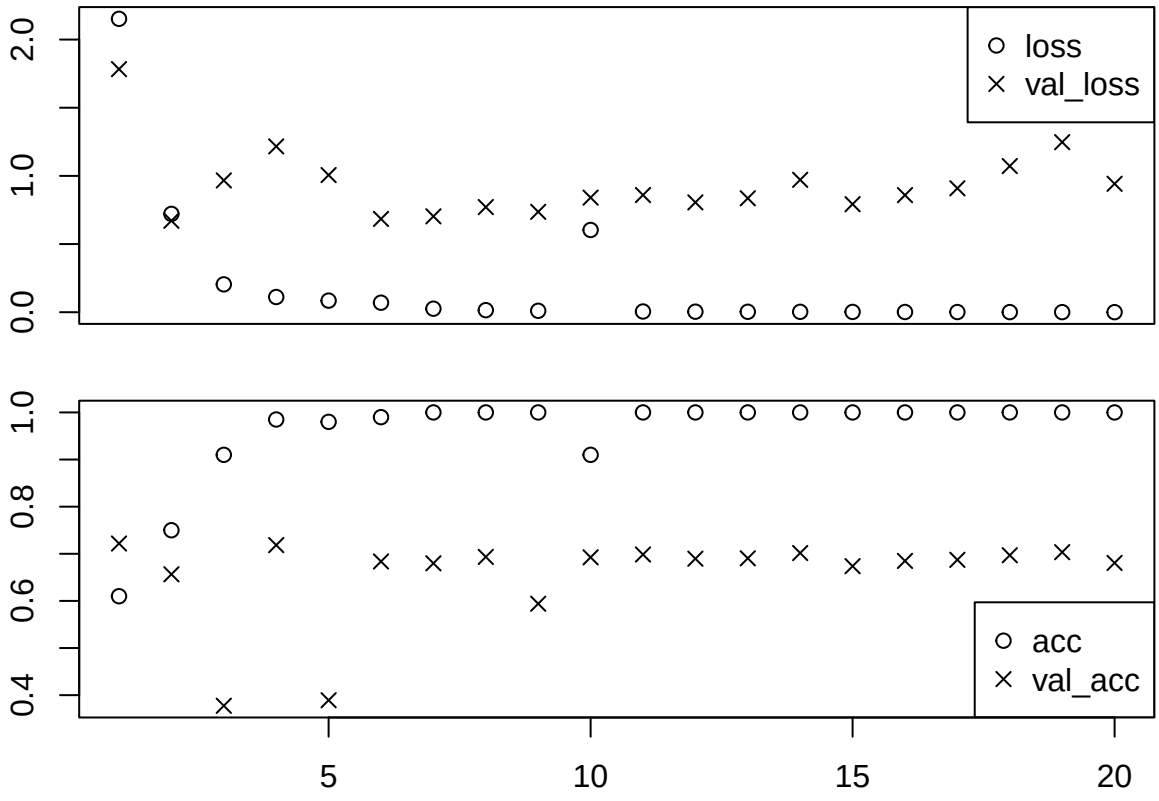
```
y_val <- labels[validation_indices]
```

```
# Embedding model with 200 training samples
model_emb <- 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_emb %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
history <- model_emb %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history)
```




```
# Pretrained model with 200 training samples
model_pre <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim, input_length = maxlen) %>%
```

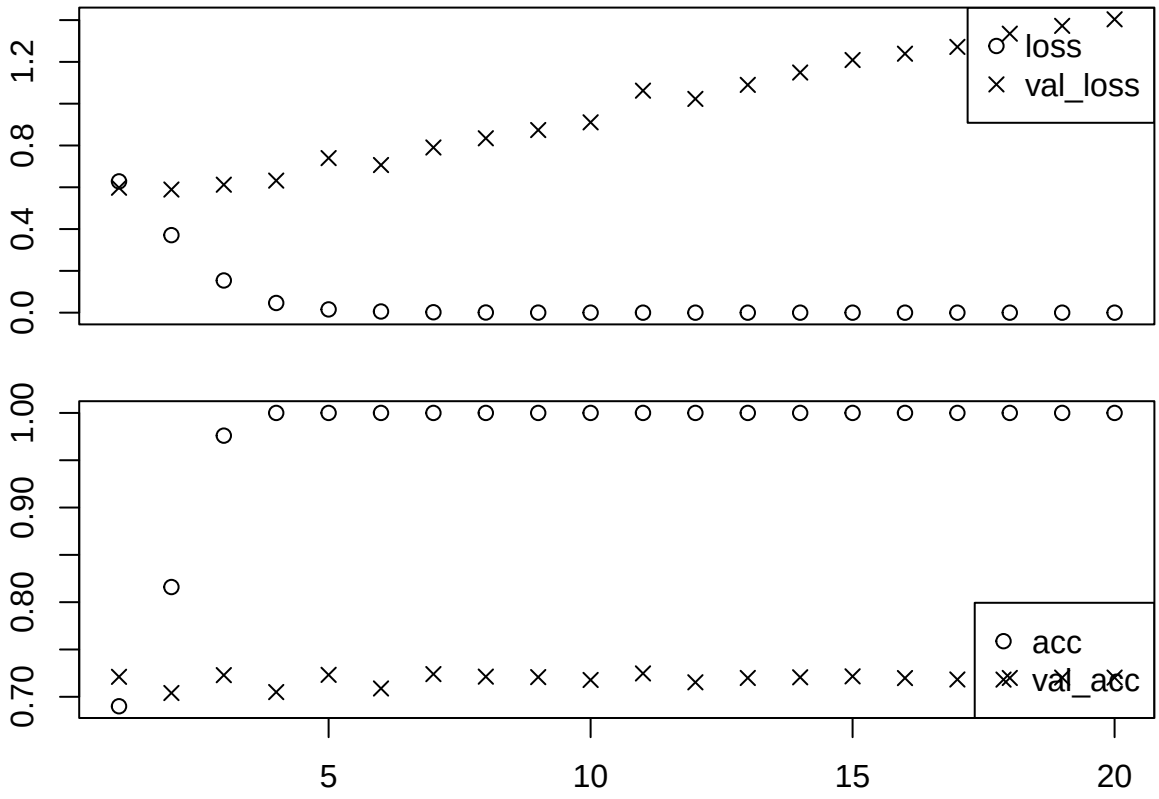
```
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

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

model_pre %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
history <- model_pre %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history)
```



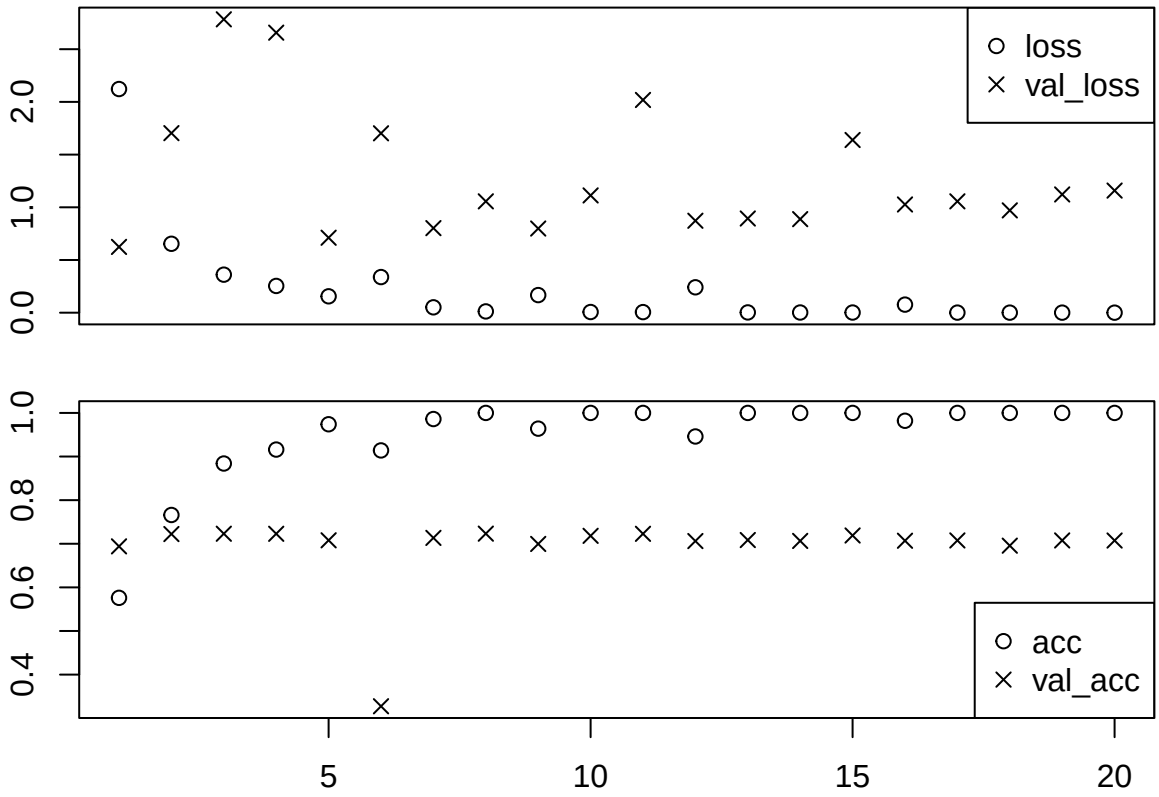
The embedding model at 200 training samples is .7234 The pretrained model at 200 training samples is .7187 The models will be run a third and final time with 500 training samples

```
training_samples <- 500
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]
```

```
# Embedding model with 500 training samples
model_emb <- 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_emb %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
history <- model_emb %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history)
```




```
# Pretrained model with 500 training samples
model_pre <- 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")

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

model_pre %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
history <- model_pre %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history)
```



The embedding model at 500 training samples is .7226 The pre-trained model at 500 training samples is .7236

In all of these, the only pre-trained model better than embedding is the 500 training sample model. The increase in sample size only slightly improved the embedding layer models. I am unsure of why the embedding model would perform worse or the pre-trained model to perform better. It's possible it was because of the only data set I was able to find only included "test" data sets.