Final Assignment: Code and Results

Ethan Luster

12/12/2021

This PDF includes the code and results of the text classification of Amazon Fine Food Reviews. As data is choosen randomly each time a model is run, my analysis of the results will be in the separate PDF named eluster_final_analysis.

This code sets-up the correct Python and Keras environment in RStudio.

```
library(reticulate)
use_condaenv('r-reticulate')
library(tensorflow)
library(keras)
tf$constant("Hellow")
```

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

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

The data comes from the website Kaggle The URL is: https://www.kaggle.com/snap/amazon-fine-food-reviews?select=Reviews.csv

This dataset contains over 500,000 food reviews from Amazon.com. The dataset includes ratings of the food on a five star scale, and the text comments that go along with the review. Positive reviews will be 4 and 5 stars, and negative will be 1-3. The goal will be to build the best model at predicting a positive or negative review based on the text.

```
maxlen <- 150 # maximum length: 150 words
max_features <- 10000 # maximum features: 10,000
max_words <- 10000 # maximum words used: 10000

amz_reviews <- read.csv("C:/Users/13303/Downloads/Reviews/Reviews.csv") %>%
```

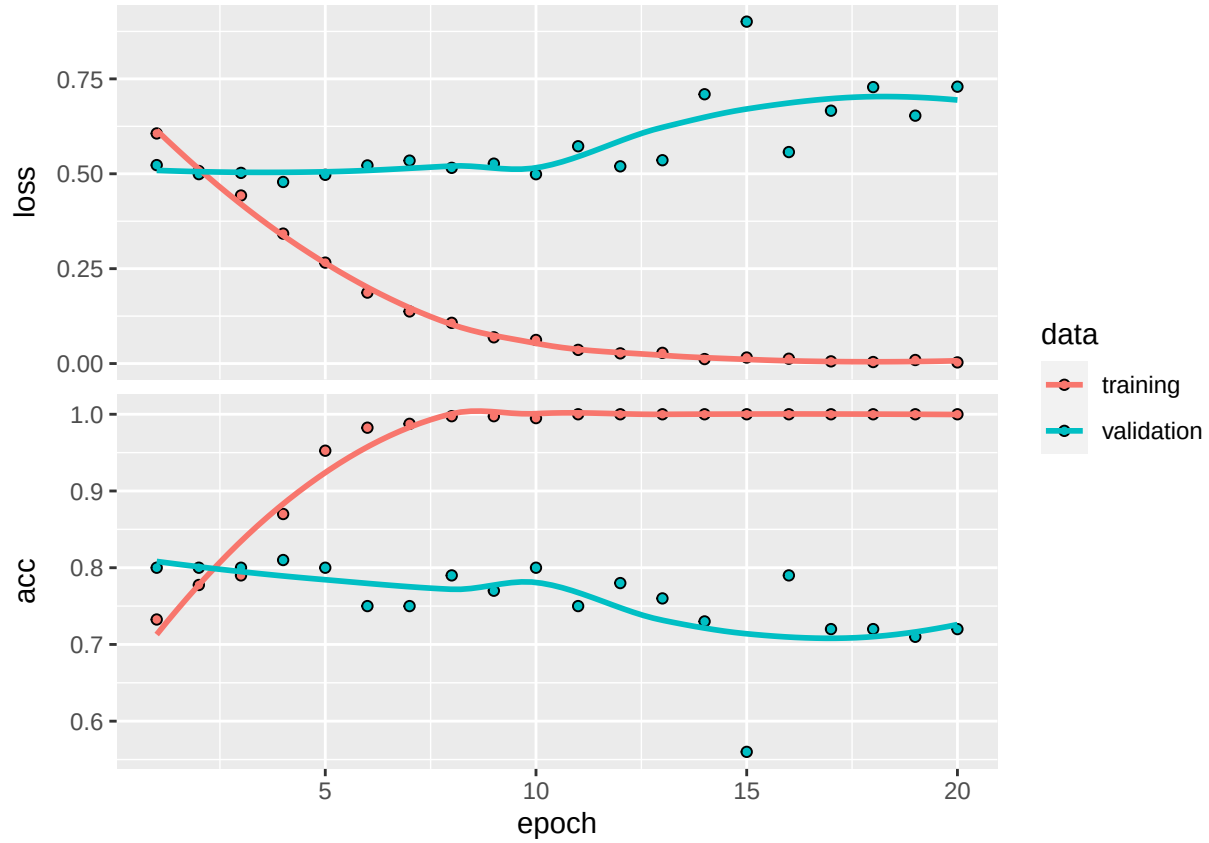
```
  mutate(Liked = ifelse(Score >= 4, 1, 0))
# Reviews are on a five star scale. If a review is a 4 or 5, it is a 1, or positive

text <- amz_reviews$Text
max_words <- 1000
tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(text)
sequences <- texts_to_sequences(tokenizer, text)
word_index = tokenizer$word_index
labels <- c()
labels <-amz_reviews$Liked
labels <- as.array(labels)
maxlen <- 100
data <- pad_sequences(sequences, maxlen = maxlen)
training_samples <- 500 # maximum training samples: 500
validation_samples <- 10000
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]
```

Model 1: Simple RNN

This model seeks to give a baseline of text classification.

```
model_simplernn <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_simple_rnn(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_simplernn %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
simple_rnn <- model_simplernn %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)

plot(simple_rnn)
```

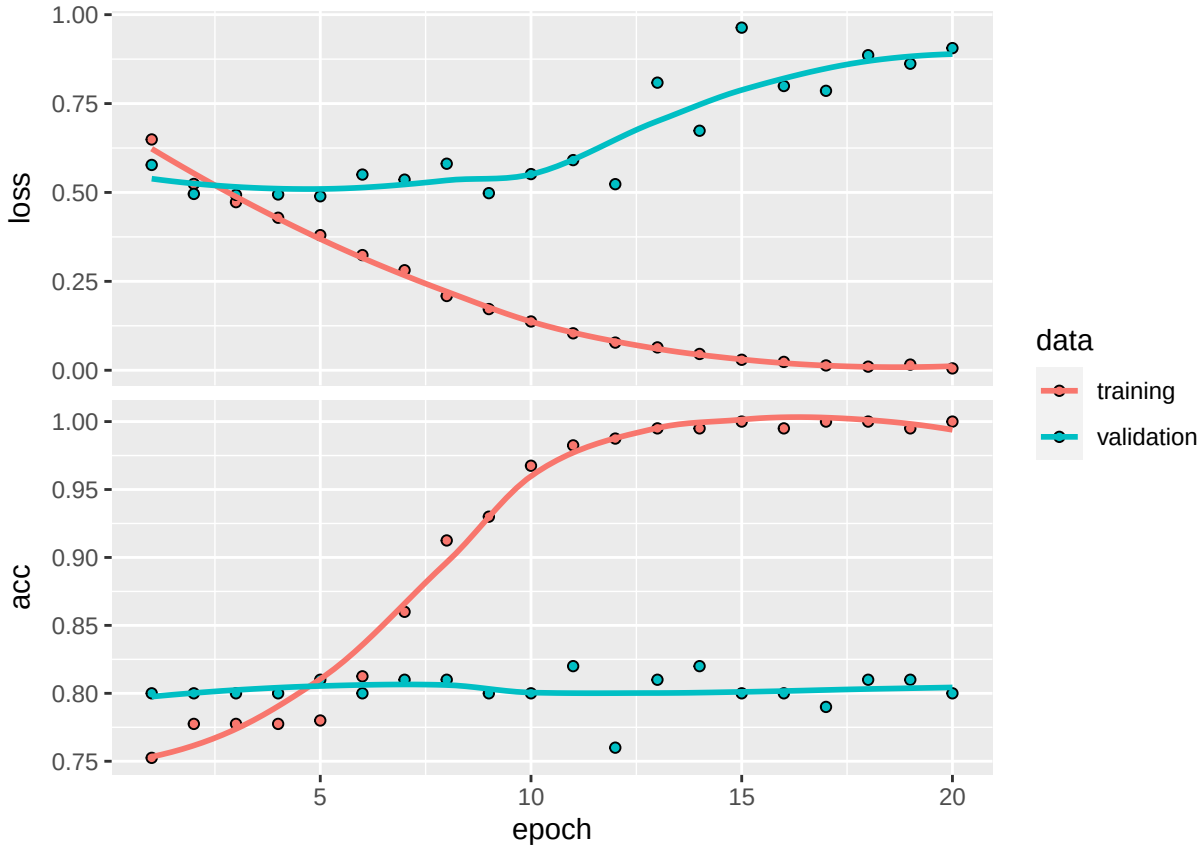
```
## `geom_smooth()` using formula 'y ~ x'
```



Two other layers exist for text classification: GRU and LSTM. I will test both of them with the same small sample size. # Model 2: GRU Layer (small sample)

```
model_g <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_gru(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_g %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
gru_model <- model_g %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(gru_model)
```

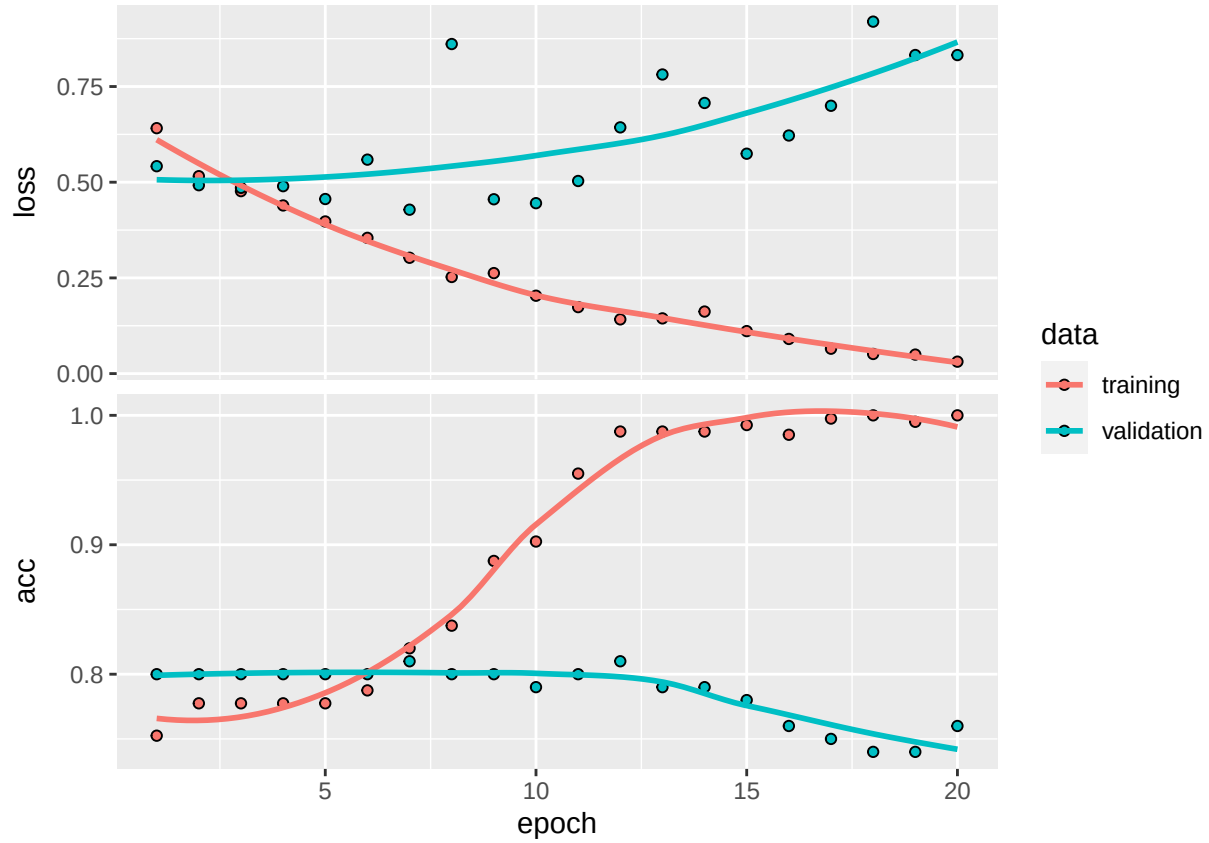
```
## `geom_smooth()` using formula 'y ~ x'
```



Model 3: LSTM Layer(small sample)

```
model_l <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_l %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
lstm_model <- model_l %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(lstm_model)
```

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



Next, I will increase sample size to 5000, and run both GRU and LSTM again

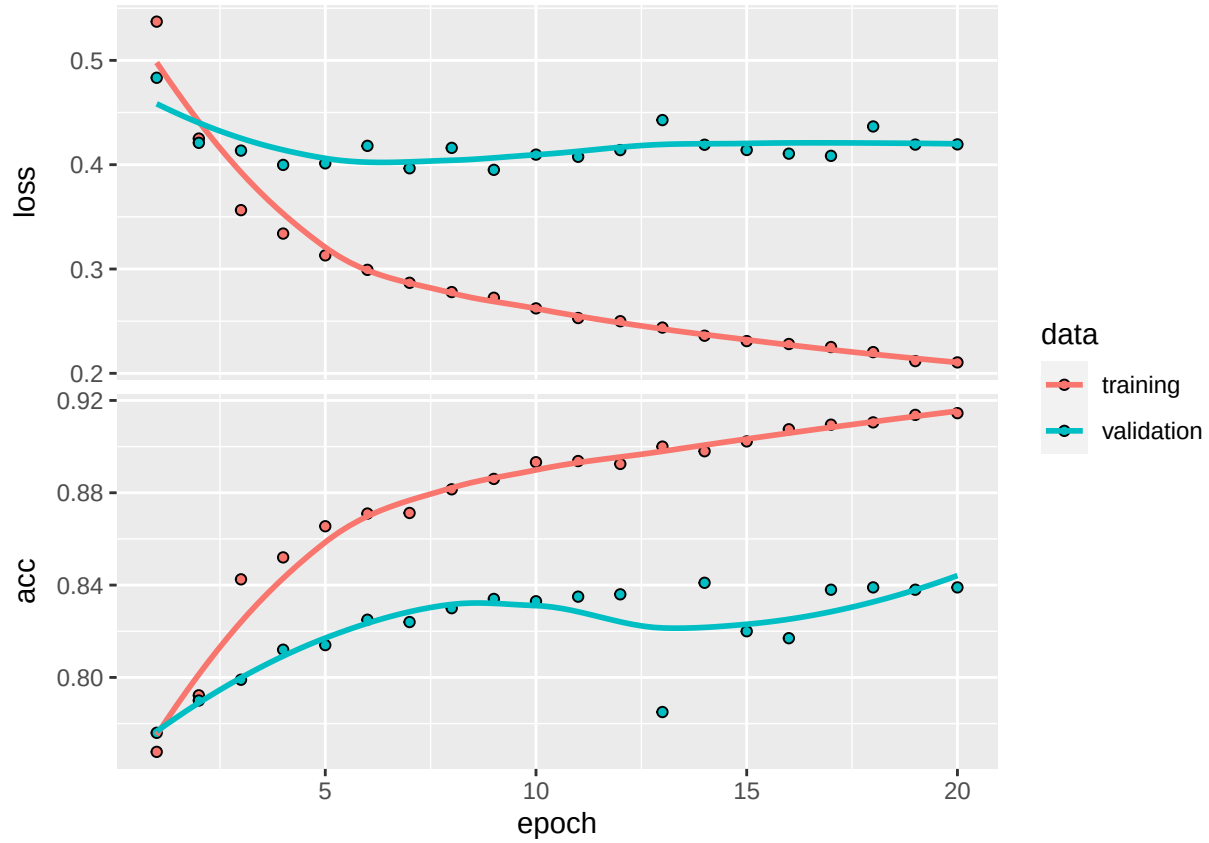
```
maxlen <- 150 # maximum length: 150 words
max_features <- 10000 # maximum features: 10,000
max_words <- 10000 # maximum words used: 10000

amz_reviews <- read.csv("C:/Users/13303/Downloads/Reviews/Reviews.csv") %>%
  mutate(Liked = ifelse(Score >= 4, 1, 0))
# Reviews are on a five star scale. If a review is a 4 or 5, it is a 1, or positive

text <- amz_reviews$Text
max_words <- 1000
tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(text)
sequences <- texts_to_sequences(tokenizer, text)
word_index = tokenizer$word_index
labels <- c()
labels <-amz_reviews$Liked
labels <- as.array(labels)
maxlen <- 100
data <- pad_sequences(sequences, maxlen = maxlen)
training_samples <- 5000 # maximum training samples: 5000
validation_samples <- 10000
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]
```

Model 4: GRU (5000 samples)

```
model_g2 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_gru(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_g2 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
gru_model2 <- model_g2 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(gru_model2)
```

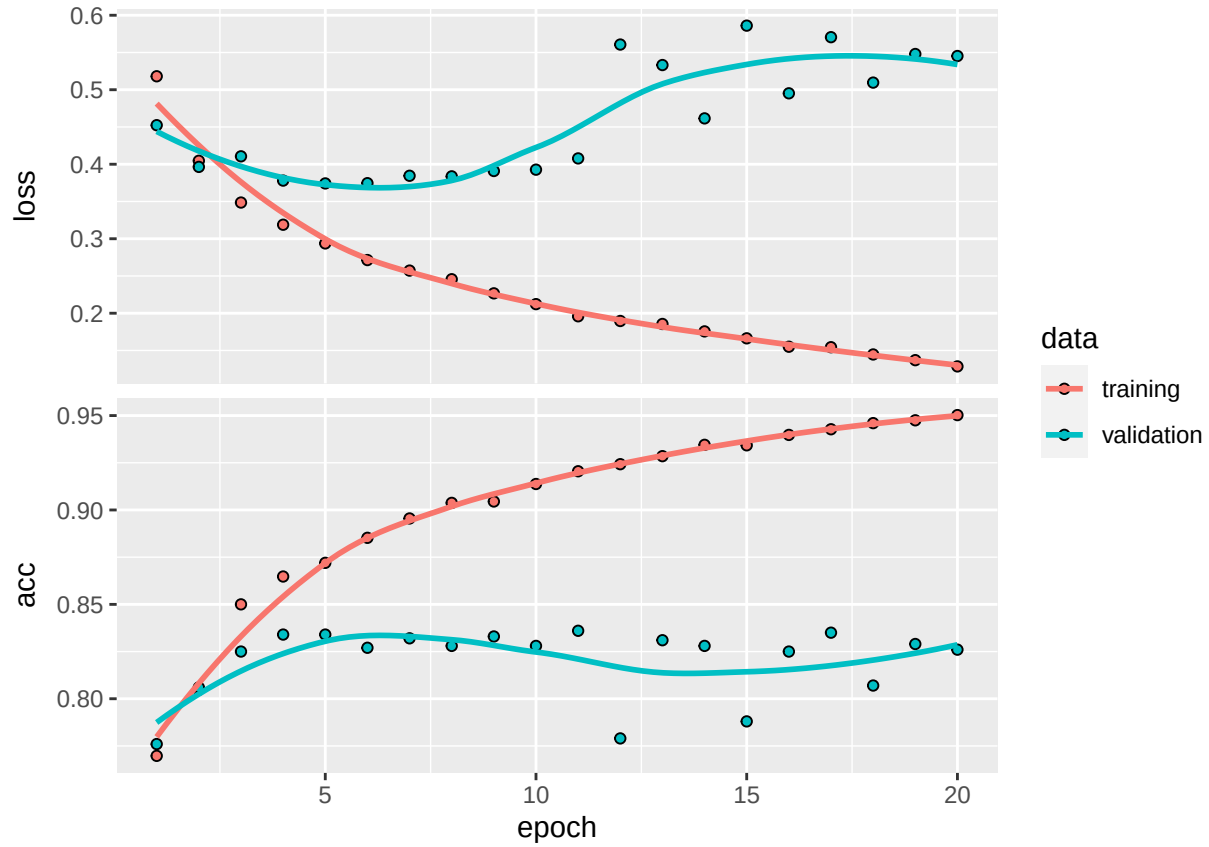
```
## `geom_smooth()` using formula 'y ~ x'
```



Model 5: LSTM (25000 samples)

```
model_l2 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_l2 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
lstm_model2 <- model_l2 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(lstm_model2)
```

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



Next, I will attempt to build a more accurate model with the LSTM layer and more samples.

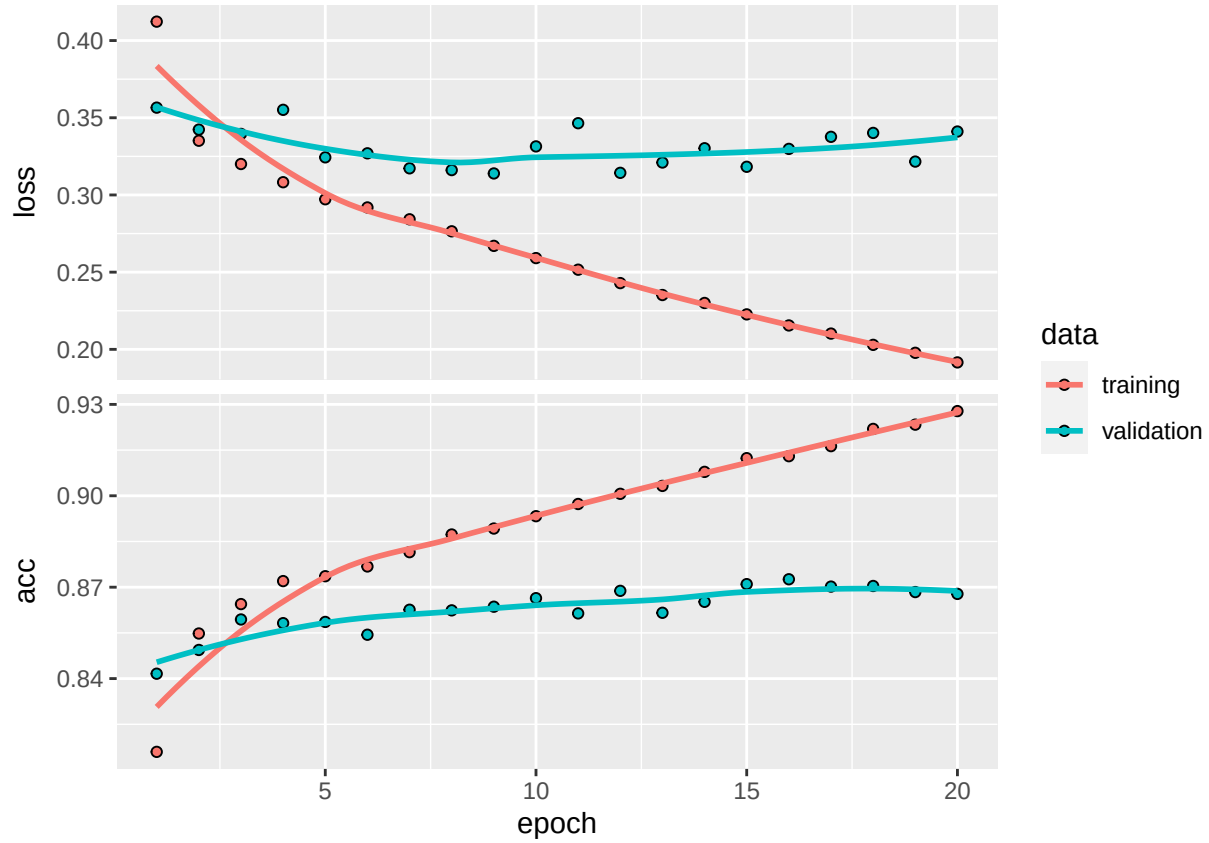
```
maxlen <- 150 # maximum length: 150 words
max_features <- 10000 # maximum features: 10,000
max_words <- 10000 # maximum words used: 10000

amz_reviews <- read.csv("C:/Users/13303/Downloads/Reviews/Reviews.csv") %>%
  mutate(Liked = ifelse(Score >= 4, 1, 0))
# Reviews are on a five star scale. If a review is a 4 or 5, it is a 1, or positive

text <- amz_reviews$Text
max_words <- 1000
tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(text)
sequences <- texts_to_sequences(tokenizer, text)
word_index = tokenizer$word_index
labels <- c()
labels <-amz_reviews$Liked
labels <- as.array(labels)
maxlen <- 100
data <- pad_sequences(sequences, maxlen = maxlen)
training_samples <- 25000 # maximum training samples: 25000
validation_samples <- 10000
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]
```

Model 6: LSTM and 25000 training samples

```
model_l3 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_l3 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
lstm_model3 <- model_l3 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(lstm_model3)
```

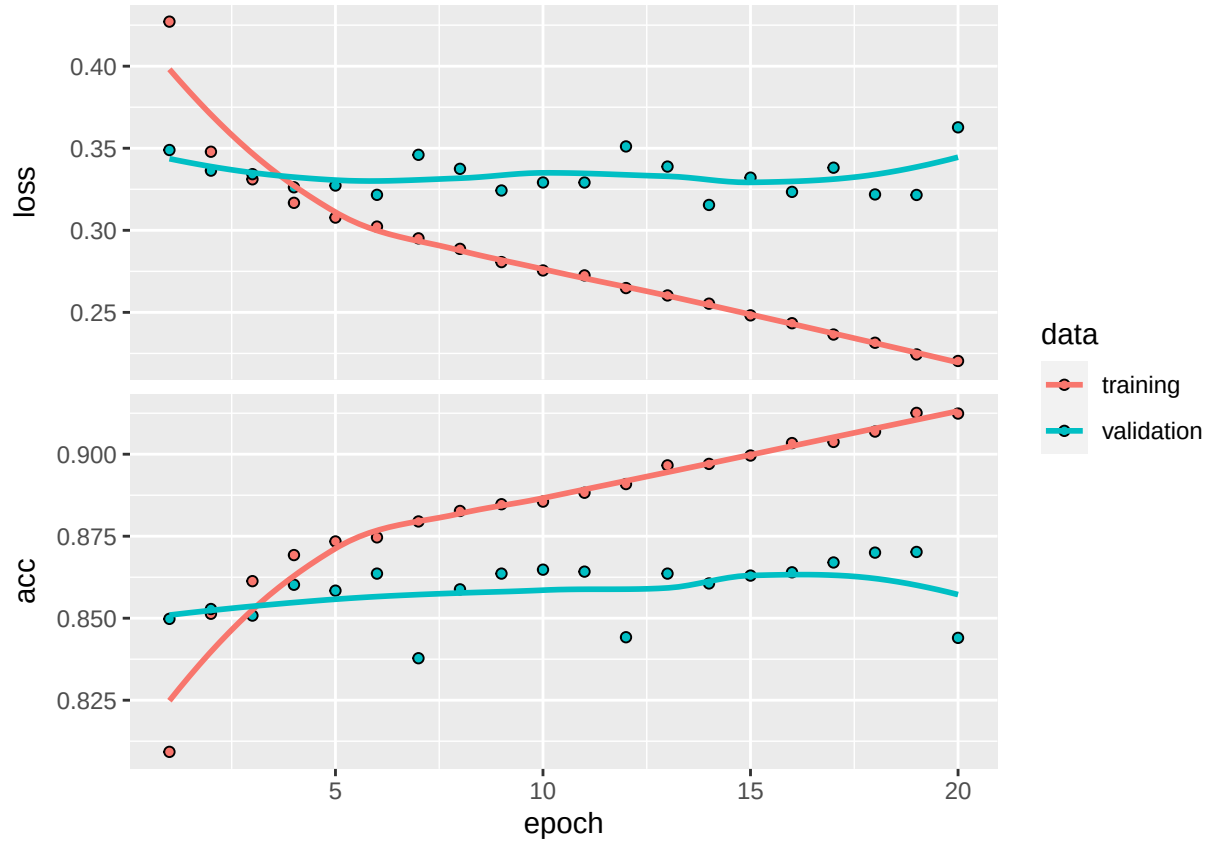
```
## `geom_smooth()` using formula 'y ~ x'
```



Model 7: LSTM plus dropout layer

```
model_l4 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_l4 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
lstm_model4 <- model_l4 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(lstm_model4)
```

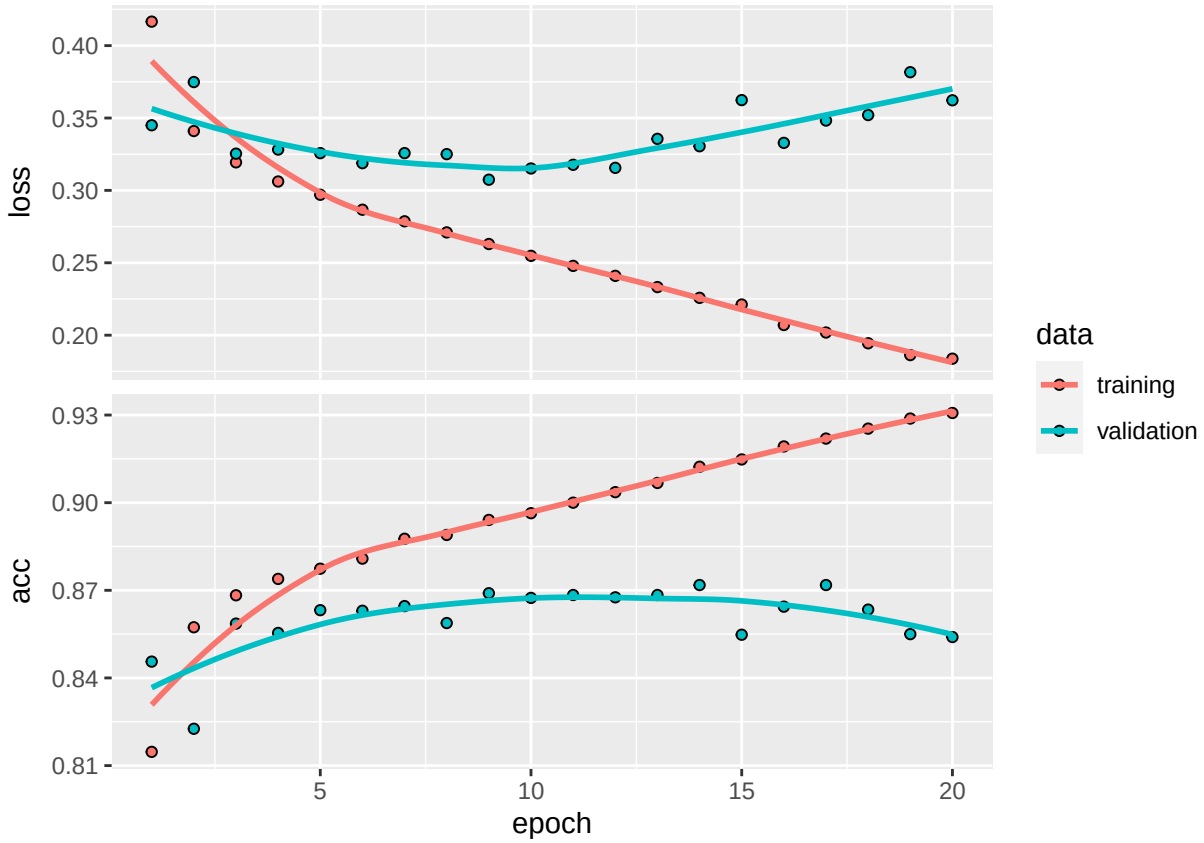
```
## `geom_smooth()` using formula 'y ~ x'
```



Model 8: LSTM, Dropout layer, and increased output_dim

```
model_l5 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 64) %>%
  layer_lstm(units = 32) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_l5 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
lstm_model5 <- model_l5 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(lstm_model5)
```

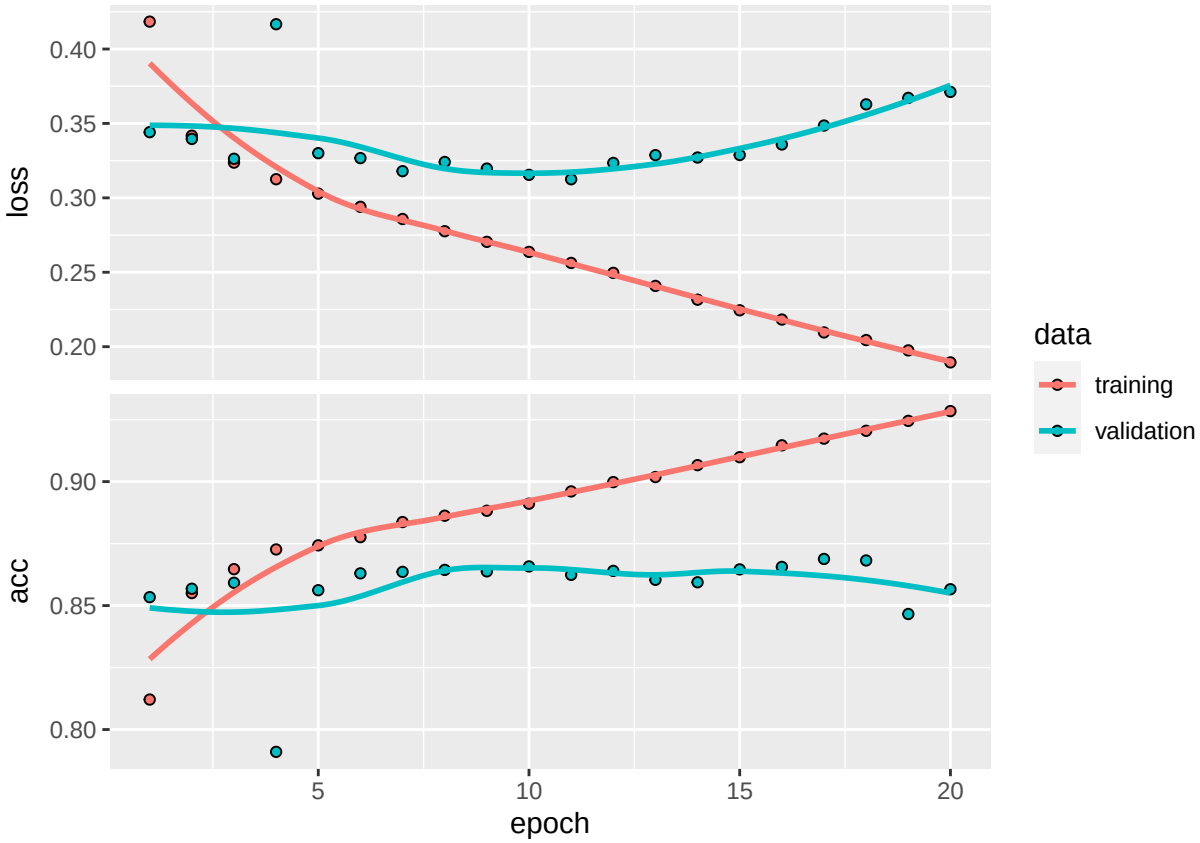
```
## `geom_smooth()` using formula 'y ~ x'
```



Model 9: LSTM and lower dropout rate

```
model_l6 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 1, activation = "sigmoid")
model_l6 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
lstm_model6 <- model_l6 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
plot(lstm_model6)
```

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


loss
0.40
0.35
0.30
0.25
0.20
acc
0.90
0.85
0.80
5
10
15
20
epoch
data
training
validation