Assignment_3_rspake1

Set up for our model for pretrained word embedding first

```
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.1.2
```

```
maxlen <- 150
training_samples <- 100
validation_samples <- 10000
max_features <- 10000

imdb_dir <- "C:/Users/rspake1/Downloads/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)
  }
}
```

Time to set the right parameters

We will be using those parameters we established earlier for validation and training size.

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

```
## Loaded Tensorflow version 2.8.0
```

```
sequences <- texts_to_sequences(tokenizer, texts)

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

```
## Found 88584 unique tokens.
```

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

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

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

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

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

```
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]
```

Preprocessing for embedding

```
glove_dir <- "C:/Users/rspake1/Downloads/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.
```

building our embedding matrix

```
embedding_dim <- 100

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

for(word in names(word_index)) {
  index <- word_index[[word]]
  if (index < max_features) {
    embedding_vector <- embeddings_index[[word]]
    if (!is.null(embedding_vector))
      embedding_matrix[index+1,] <- embedding_vector
  }
}
```

Defining our pre-trained model

```
model1 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

Loading GloVe into our model

```
get_layer(model1, index = 1) %>%
  set_weights(list(embedding_matrix)) %>%
  freeze_weights()
```

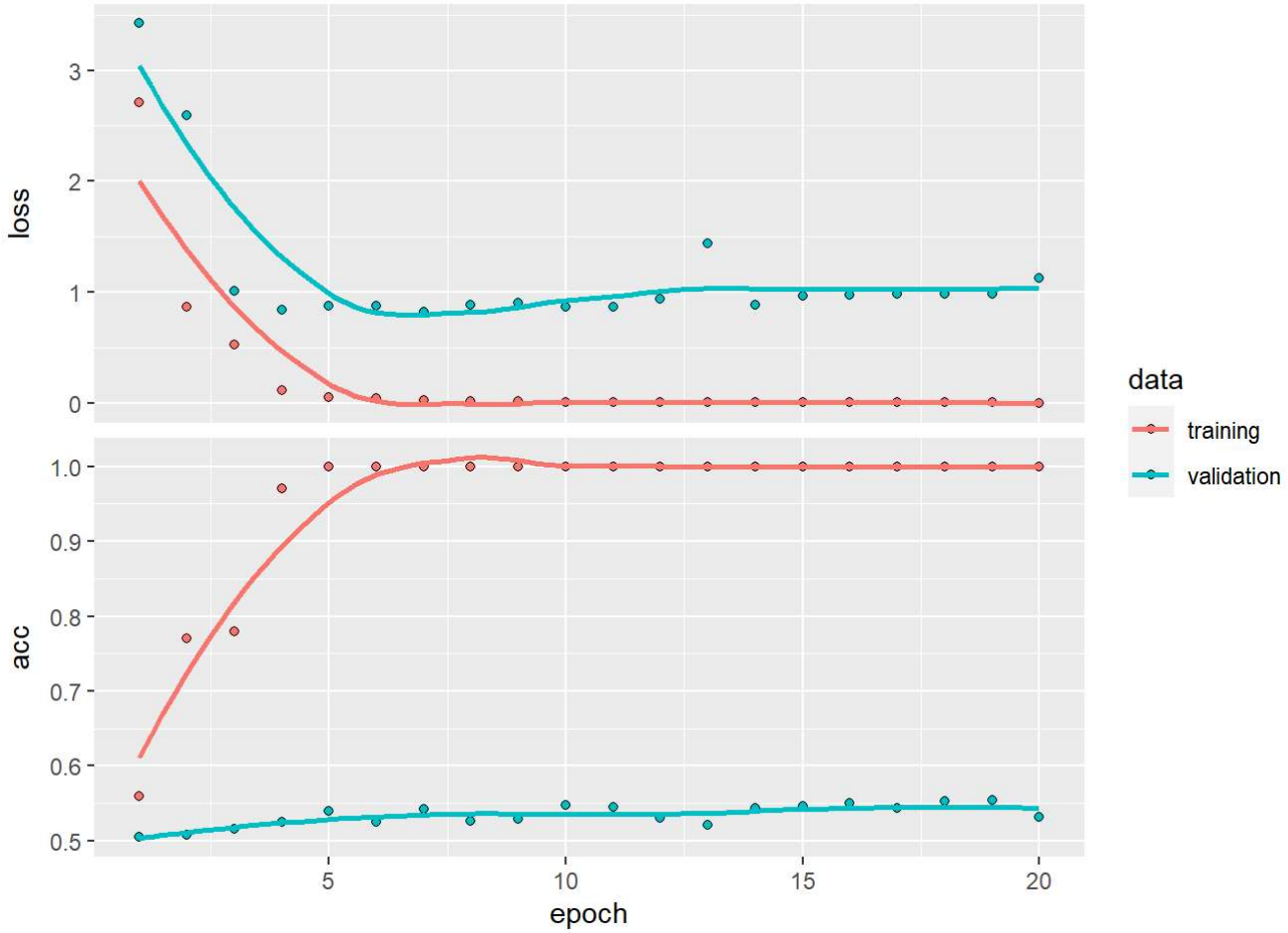
Training/Evaluation

```
model1 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history1 <- model1 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)

save_model_weights_hdf5(model1, "pre_trained_glove-model1.h5")
```

Lastly, we will plot this history so that we can compare it to the next model we make.

```
plot(history1)
```

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



As we can see here, this model overfits (basically) instantly. This is not very surprising as the training sample is very small and we already know that overfitting is bound to happen at small sample sizes. Our performance here isn't great as we hit the bend at **56%** accuracy.

Building an embedding layer ourselves

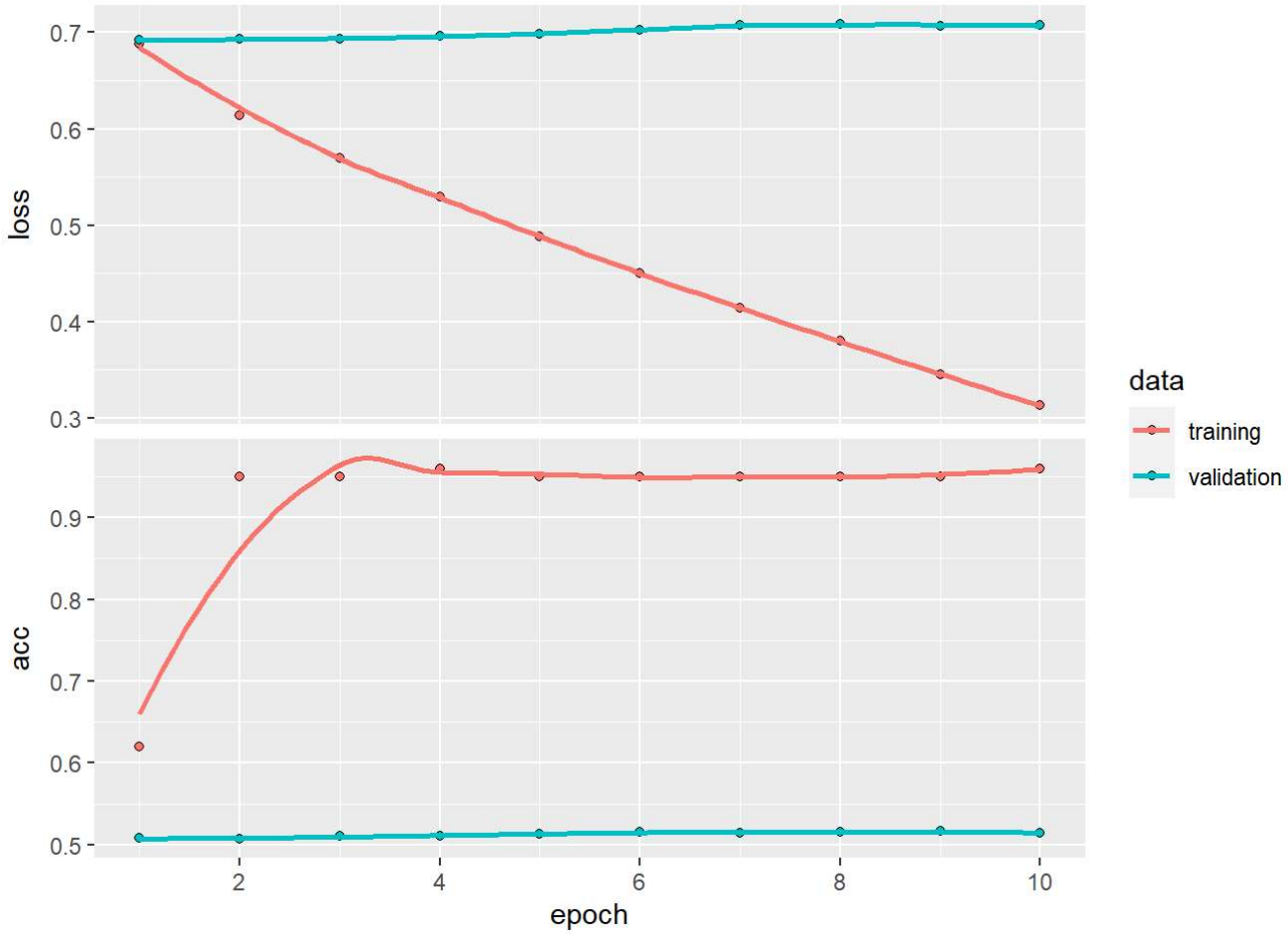
now we will build an embedding layer ourselves instead of relying on a pre-trained set.

```
model2 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 1, activation = "sigmoid")

model2 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history2 <- model2 %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)

save_model_weights_hdf5(model2, "assignment_3_model2.h5")
```

Lastly, lets plot our results from this model.

```
plot(history2)
```

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



Unfortunately, this model does not perform any better than the previous one. With the best accuracy coming out to **51%**

Optimizing our model

Lets adjust our training sample size to see at what point our embedded layer model can perform better.

```
training_samples2 <- 1500


training_indices2 <- indices[1:training_samples2]
validation_indices2 <- indices[(training_samples2 + 1):
                               (training_samples2 + validation_samples)]

x_train2 <- data[training_indices2,]
y_train2 <- labels[training_indices2]

x_val2 <- data[validation_indices2,]
y_val2 <- labels[validation_indices2]
```

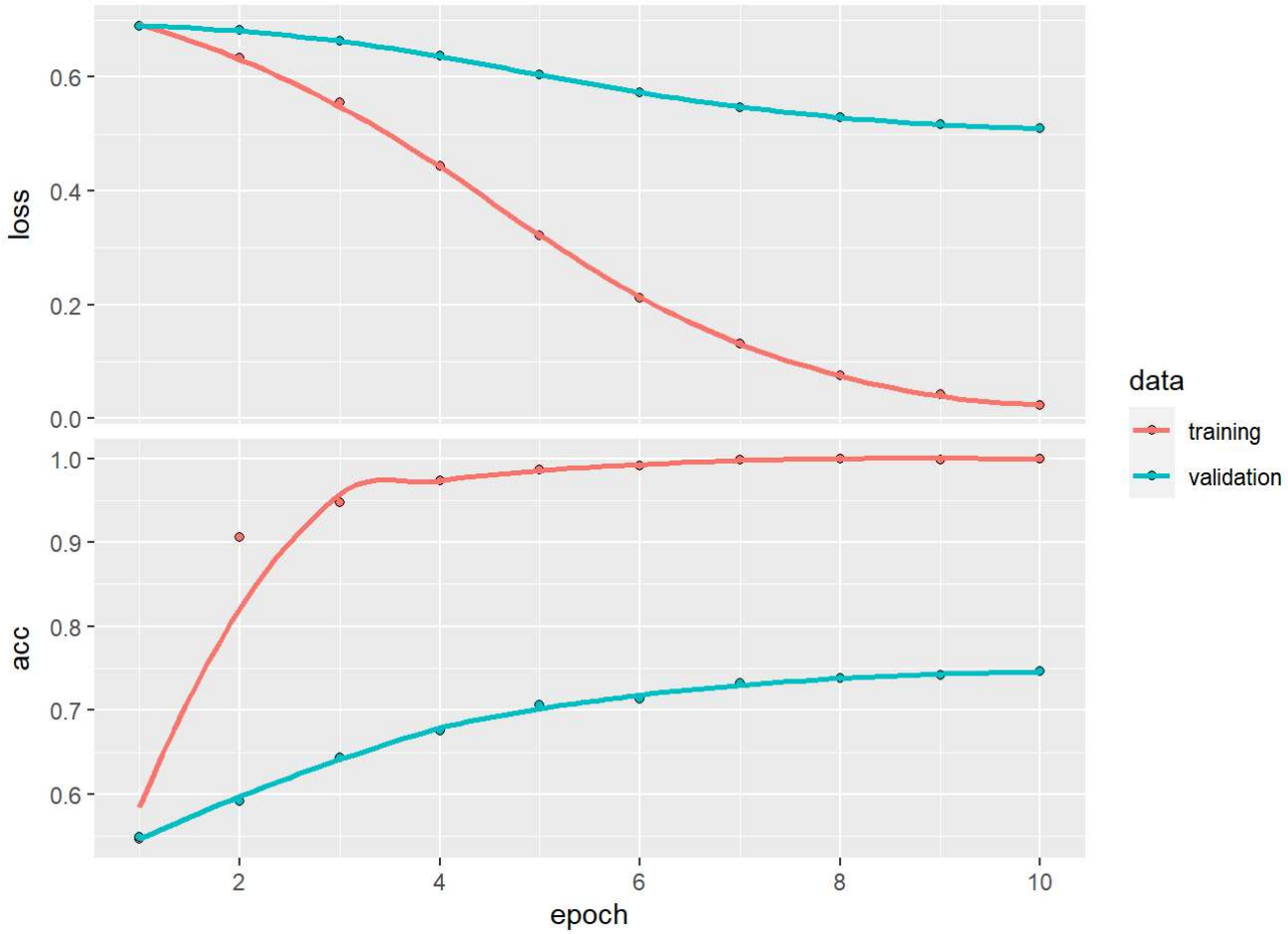
We have changed the sample size of the training data from **100** units to **1500** units and will see how changing JUST that effects our model.

```
model3 <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 1, activation = "sigmoid")

model3 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

history3 <- model3 %>% fit(
  x_train2, y_train2,
  epochs = 10,
  batch_size = 32,
  validation_data = list(x_val2, y_val2)
)

save_model_weights_hdf5(model2, "assignment_3_model3.h5")
```

finally lets see this new model graphically.

```
plot(history3)
```

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



With this new model, our best performance occurs around epoch 7, with a val_acc of **71%** and a val_loss of **56%** which is much better than the pre-trained model. As we continue to increase the size of the training data, we will continue to see an improvement in the performance of the embedded model.

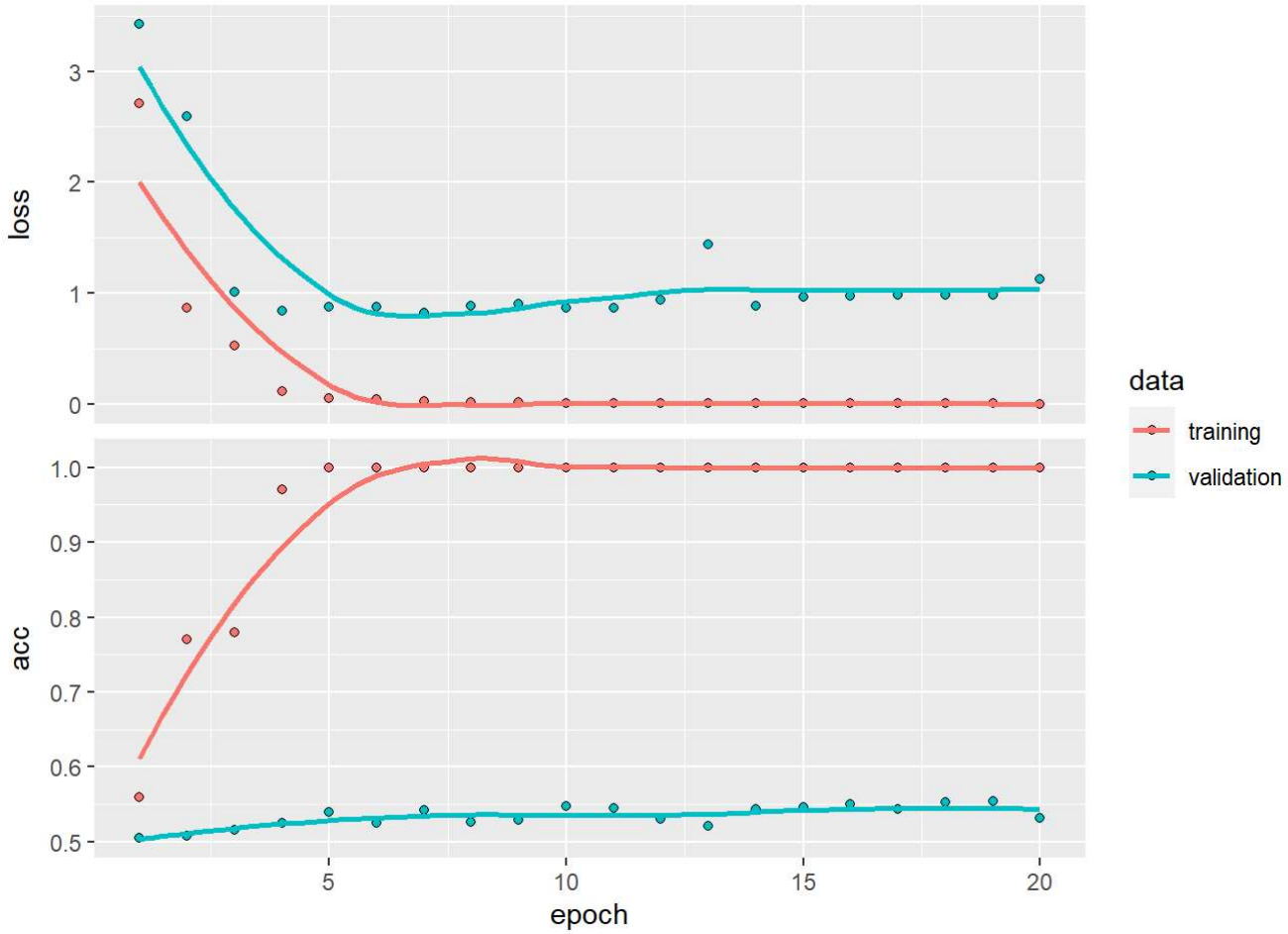
New Sample size through pre-Trained Model

Bellow we will put our new sample size through our pre-trained model and see how it is effected, then compare our two new outputs.

```
history4 <- model1 %>% fit(
  x_train2, y_train2,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val2, y_val2)
)
```

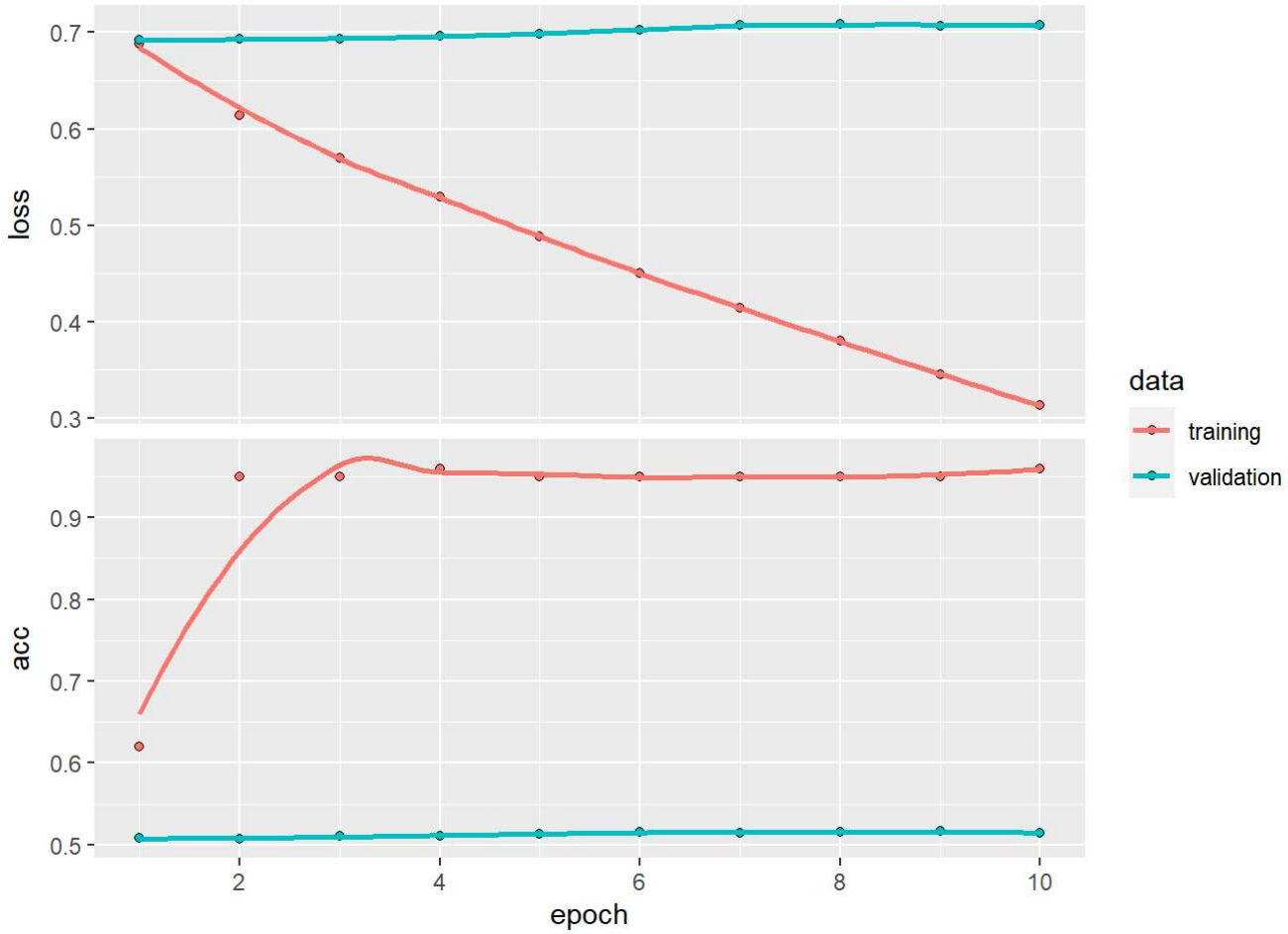
```
plot(history1)
```

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



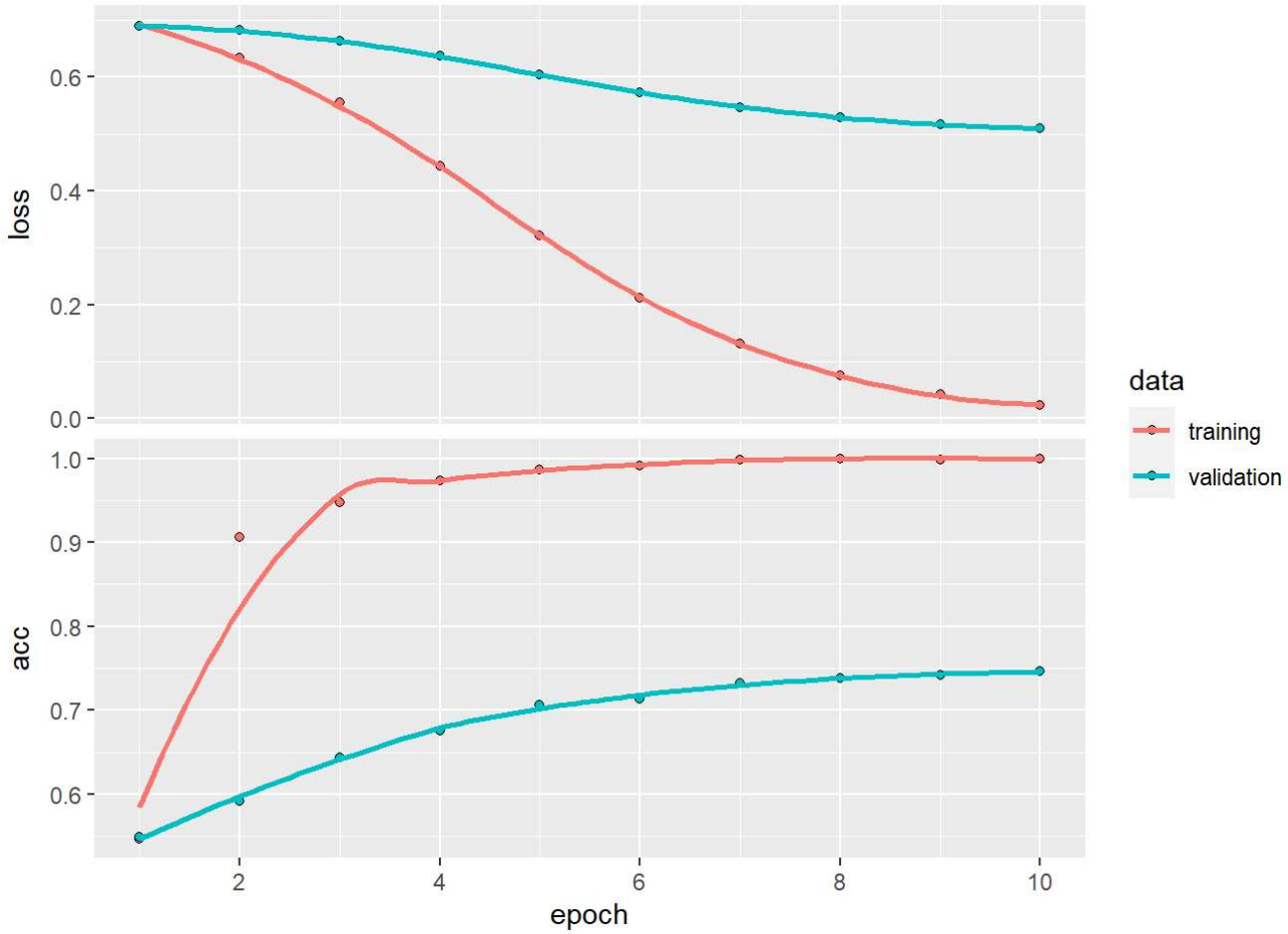

```
plot(history2)
```

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



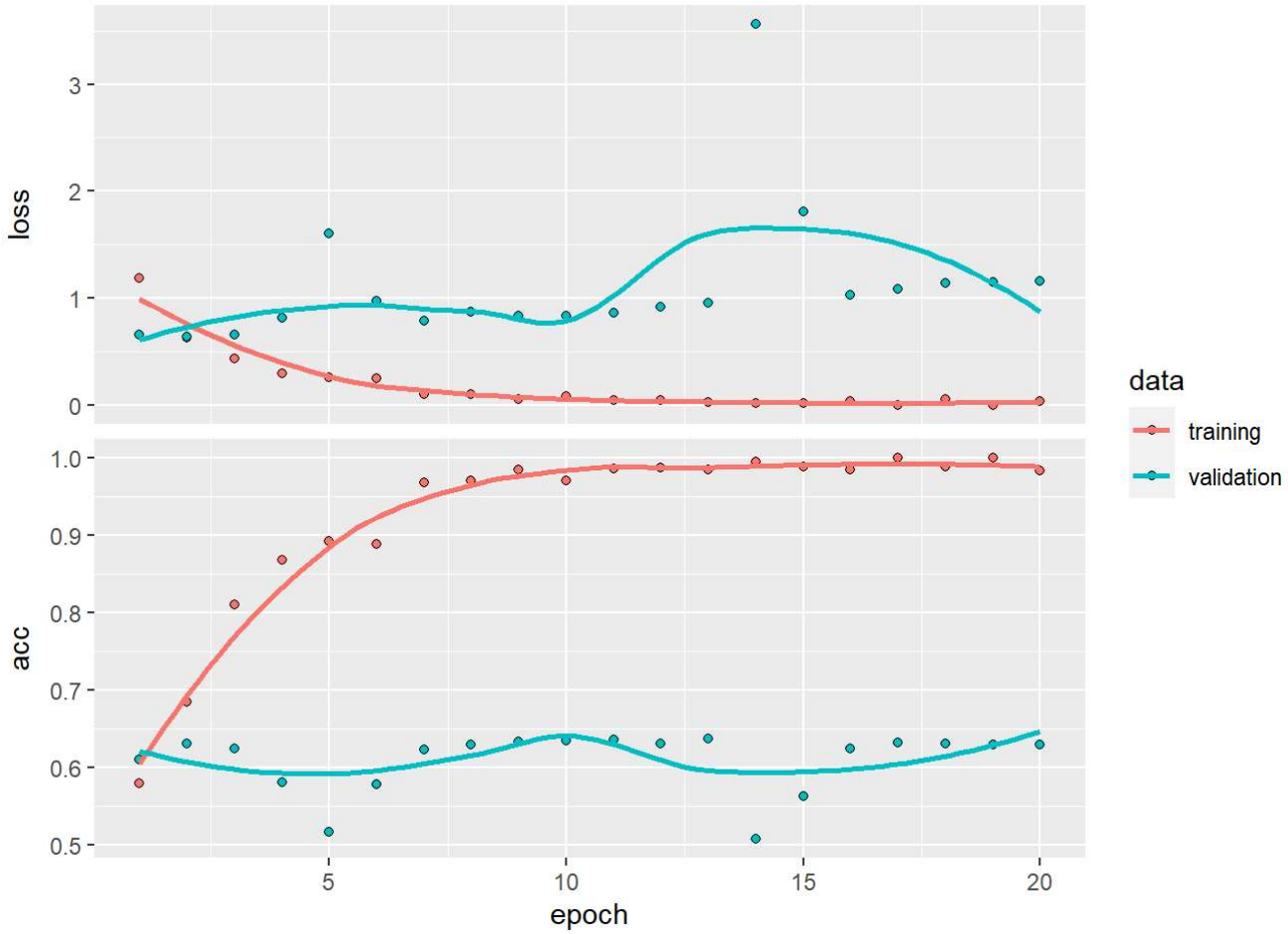

```
plot(history3)
```

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




```
plot(history4)
```

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



Further Proof

To be doubly sure that the embedding model performs better at this sample size, I have added another plot where I applied the new sample size to the pre-trained model as well and have displayed the results above.