bshield6\_Assignment\_3

Brandon Lloyd Shields

11/14/2021

library(keras)  
library (dplyr)

## Registered S3 methods overwritten by 'tibble':  
## method from   
## format.tbl pillar  
## print.tbl pillar

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

#Importing dataset from web download.   
  
imdb\_dir <- "~/Downloads/aclImdb"  
train\_dir <- file.path(imdb\_dir, "train")  
  
labels <- c()  
texts <- c()  
  
#Creating Labeled dataset for traibning and validating  
  
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)  
 }  
}

#Setting parameters based on assignment instructions.  
  
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 88582 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

# Split the data into a training set and a validation set and shuffeling the data  
# since samples are ordered  
  
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]

#activating pre-trained layers to include in model  
  
glove\_dir = '~/Downloads/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")

## 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))  
 # Words not found in the embedding index will be all zeros.  
 embedding\_matrix[index+1,] <- embedding\_vector  
 }  
}

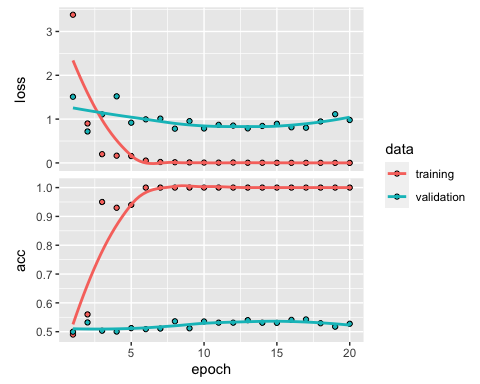
model <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = embedding\_dim,   
 input\_length = maxlen) %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 32, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")

get\_layer(model, index = 1) %>%   
 set\_weights(list(embedding\_matrix)) %>%   
 freeze\_weights()

model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.pt.100 <- model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val, y\_val)  
)  
  
save\_model\_weights\_hdf5(model, "pre\_trained\_glove\_model.h5")

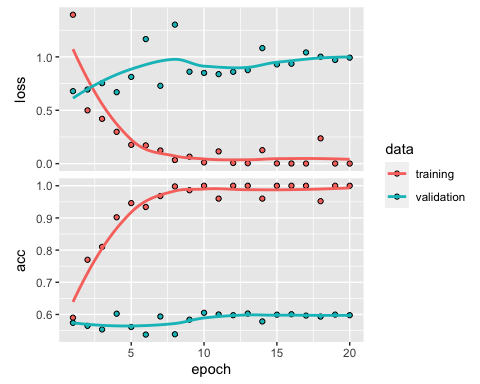
plot(history.pt.100)

## `geom\_smooth()` using formula 'y ~ x'



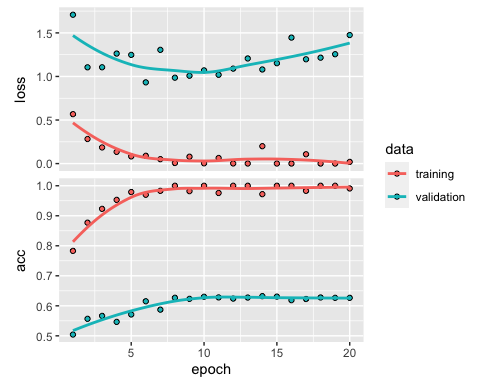
#Crearibg various training Samples Sizes for all future models  
training\_samples\_500 <- 500  
training\_samples\_1000 <- 1000  
training\_samples\_1500 <- 1500  
training\_samples\_2000 <- 2000  
  
#Indicies with 500 training set  
indices\_500 <- sample(1:nrow(data))  
training\_indices\_500 <- indices[1:training\_samples\_500]  
validation\_indices\_500 <- indices[(training\_samples\_500 + 1):   
 (training\_samples\_500 + validation\_samples)]  
  
x\_train\_500 <- data[training\_indices\_500,]  
y\_train\_500 <- labels[training\_indices\_500]  
  
x\_val\_500 <- data[validation\_indices\_500,]  
y\_val\_500 <- labels[validation\_indices\_500]  
  
model.pt.500 <- 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.pt.500, index = 1) %>%   
 set\_weights(list(embedding\_matrix)) %>%   
 freeze\_weights()  
  
model.pt.500 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.pt.500 <- model %>% fit(  
 x\_train\_500, y\_train\_500,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_500, y\_val\_500)  
)  
  
save\_model\_weights\_hdf5(model, "pre\_trained\_glove\_model.h5\_pt500")  
  
  
plot(history.pt.500)

## `geom\_smooth()` using formula 'y ~ x'



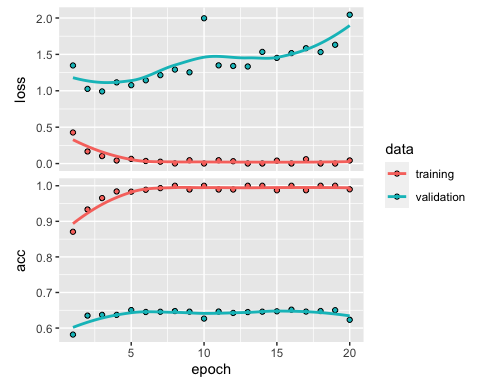
#Indicies with 1000 training set  
  
indices\_1000 <- sample(1:nrow(data))  
training\_indices\_1000 <- indices[1:training\_samples\_1000]  
validation\_indices\_1000 <- indices[(training\_samples\_1000 + 1):   
 (training\_samples\_1000 + validation\_samples)]  
  
x\_train\_1000 <- data[training\_indices\_1000,]  
y\_train\_1000 <- labels[training\_indices\_1000]  
  
x\_val\_1000 <- data[validation\_indices\_1000,]  
y\_val\_1000 <- labels[validation\_indices\_1000]  
  
model.pt.1000 <- 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.pt.1000, index = 1) %>%   
 set\_weights(list(embedding\_matrix)) %>%   
 freeze\_weights()  
  
model.pt.1000 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.pt.1000 <- model %>% fit(  
 x\_train\_1000, y\_train\_1000,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_1000, y\_val\_1000)  
)  
  
save\_model\_weights\_hdf5(model, "pre\_trained\_glove\_model.h5\_pt1000")  
  
  
plot(history.pt.1000)

## `geom\_smooth()` using formula 'y ~ x'



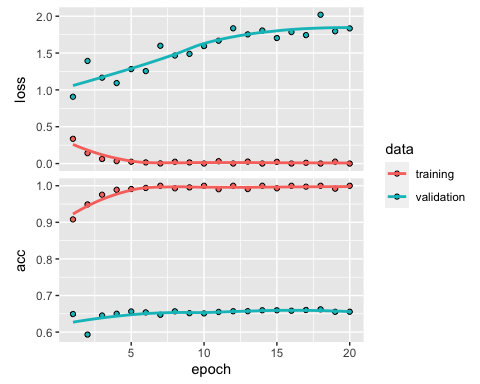
#Indicies with 1500 training set  
  
indices\_1500 <- sample(1:nrow(data))  
training\_indices\_1500 <- indices[1:training\_samples\_1500]  
validation\_indices\_1500 <- indices[(training\_samples\_1500 + 1):   
 (training\_samples\_1500 + validation\_samples)]  
  
x\_train\_1500 <- data[training\_indices\_1500,]  
y\_train\_1500 <- labels[training\_indices\_1500]  
  
x\_val\_1500 <- data[validation\_indices\_1500,]  
y\_val\_1500 <- labels[validation\_indices\_1500]  
  
model.pt.1500 <- 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.pt.1500, index = 1) %>%   
 set\_weights(list(embedding\_matrix)) %>%   
 freeze\_weights()  
  
model.pt.1500 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.pt.1500 <- model %>% fit(  
 x\_train\_1500, y\_train\_1500,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_1500, y\_val\_1500)  
)  
  
save\_model\_weights\_hdf5(model, "pre\_trained\_glove\_model.h5\_pt1500")  
  
  
plot(history.pt.1500)

## `geom\_smooth()` using formula 'y ~ x'



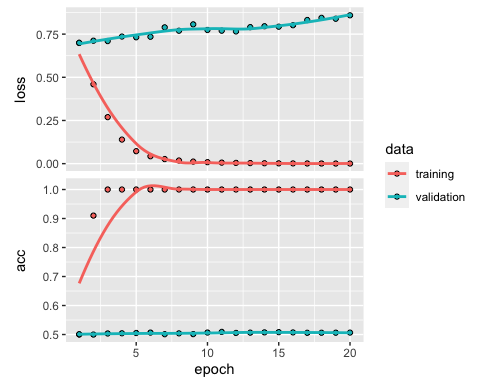
#Indicies with 2000 training set  
  
indices\_2000 <- sample(1:nrow(data))  
training\_indices\_2000 <- indices[1:training\_samples\_2000]  
validation\_indices\_2000 <- indices[(training\_samples\_2000 + 1):   
 (training\_samples\_2000 + validation\_samples)]  
  
x\_train\_2000 <- data[training\_indices\_2000,]  
y\_train\_2000 <- labels[training\_indices\_2000]  
  
x\_val\_2000 <- data[validation\_indices\_2000,]  
y\_val\_2000 <- labels[validation\_indices\_2000]  
  
model.pt.2000 <- 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.pt.2000, index = 1) %>%   
 set\_weights(list(embedding\_matrix)) %>%   
 freeze\_weights()  
  
model.pt.2000 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.pt.2000 <- model %>% fit(  
 x\_train\_2000, y\_train\_2000,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_2000, y\_val\_2000)  
)  
  
save\_model\_weights\_hdf5(model, "pre\_trained\_glove\_model.h5\_pt2000")  
  
  
plot(history.pt.2000)

## `geom\_smooth()` using formula 'y ~ x'



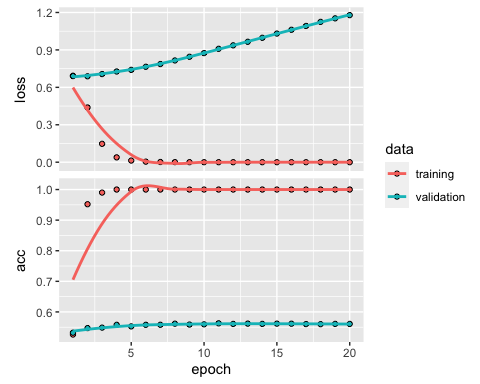
model.100 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = embedding\_dim,   
 input\_length = maxlen) %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 32, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.100 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.100 <- model.100 %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history.100)

## `geom\_smooth()` using formula 'y ~ x'



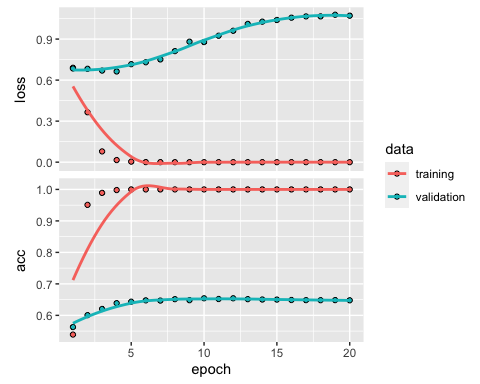
model.500 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = embedding\_dim,   
 input\_length = maxlen) %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 32, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.500 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.500 <- model.500 %>% fit(  
 x\_train\_500, y\_train\_500,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_500, y\_val\_500)  
)  
  
plot(history.500)

## `geom\_smooth()` using formula 'y ~ x'



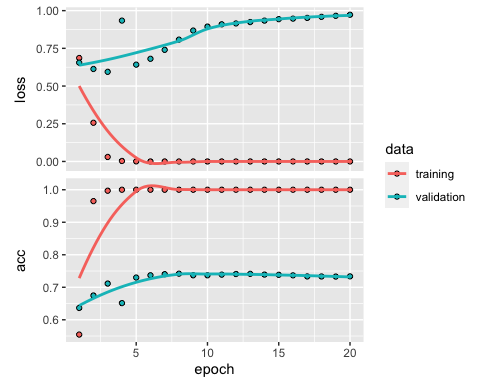
model.1000 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = embedding\_dim,   
 input\_length = maxlen) %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 32, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.1000 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.1000 <- model.1000 %>% fit(  
 x\_train\_1000, y\_train\_1000,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_1000, y\_val\_1000)  
)  
  
plot(history.1000)

## `geom\_smooth()` using formula 'y ~ x'



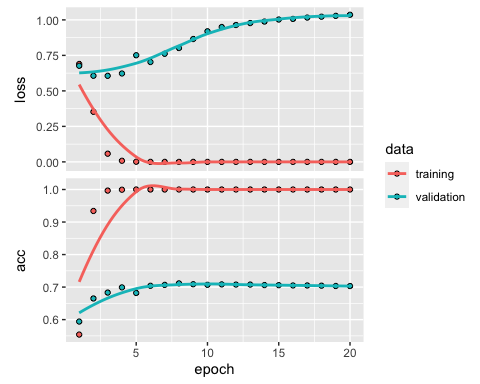
model.2000 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = embedding\_dim,   
 input\_length = maxlen) %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 32, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.2000 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.2000 <- model.2000 %>% fit(  
 x\_train\_2000, y\_train\_2000,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_2000, y\_val\_2000)  
)  
  
plot(history.2000)

## `geom\_smooth()` using formula 'y ~ x'



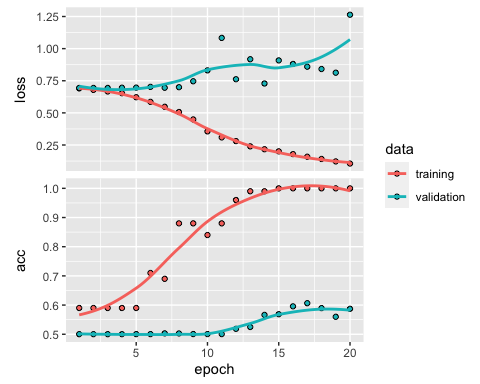
model.1500 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = embedding\_dim,   
 input\_length = maxlen) %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 32, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.1500 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("acc")  
)  
  
history.1500 <- model.1500 %>% fit(  
 x\_train\_1500, y\_train\_1500,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_1500, y\_val\_1500)  
)  
  
plot(history.1500)

## `geom\_smooth()` using formula 'y ~ x'



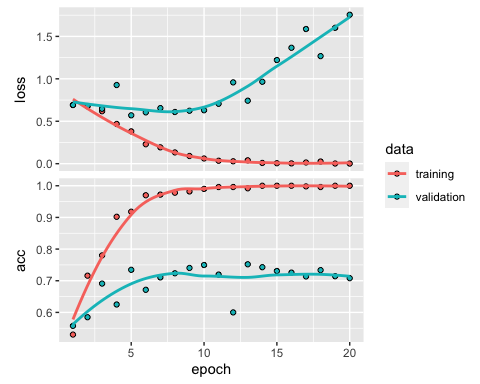
model.rnn.100 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = 32) %>%   
 layer\_lstm(units = 32) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.rnn.100 %>% compile(  
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = c("acc")  
)  
  
history.rnn.100 <- model.rnn.100 %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history.rnn.100)

## `geom\_smooth()` using formula 'y ~ x'



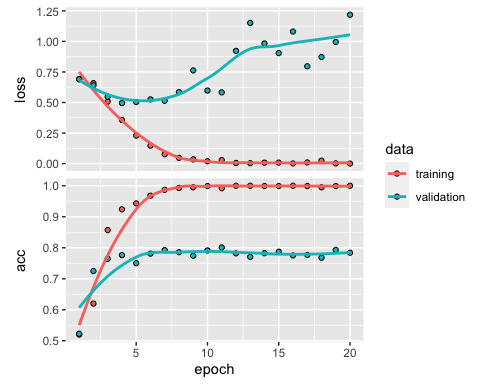
model.rnn.500 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = 32) %>%   
 layer\_lstm(units = 32) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.rnn.500 %>% compile(  
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = c("acc")  
)  
  
history.rnn.500 <- model.rnn.500 %>% fit(  
 x\_train\_500, y\_train\_500,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_500, y\_val\_500)  
)  
  
plot(history.rnn.500)

## `geom\_smooth()` using formula 'y ~ x'



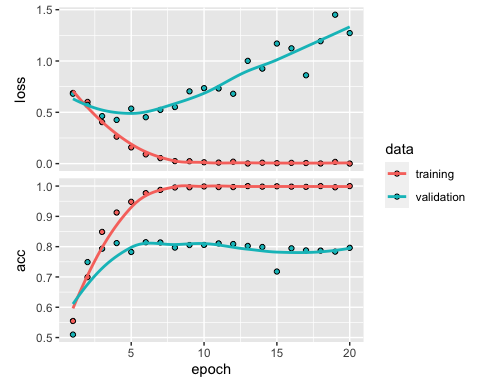
model.rnn.1000 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = 32) %>%   
 layer\_lstm(units = 32) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.rnn.1000 %>% compile(  
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = c("acc")  
)  
  
history.rnn.1000 <- model.rnn.1000 %>% fit(  
 x\_train\_1000, y\_train\_1000,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_1000, y\_val\_1000)  
)  
  
plot(history.rnn.1000)

## `geom\_smooth()` using formula 'y ~ x'



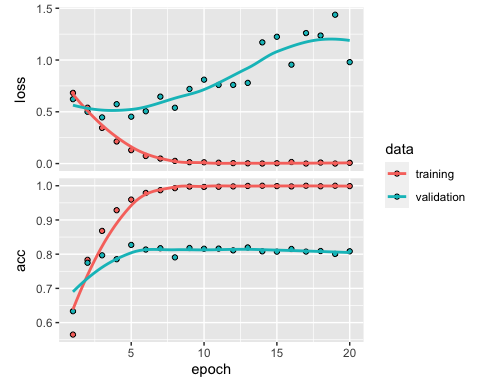
model.rnn.1500 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = 32) %>%   
 layer\_lstm(units = 32) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.rnn.1500 %>% compile(  
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = c("acc")  
)  
  
history.rnn.1500 <- model.rnn.1500 %>% fit(  
 x\_train\_1500, y\_train\_1500,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_1500, y\_val\_1500)  
)  
  
plot(history.rnn.1500)

## `geom\_smooth()` using formula 'y ~ x'



model.rnn.2000 <- keras\_model\_sequential() %>%   
 layer\_embedding(input\_dim = max\_words, output\_dim = 32) %>%   
 layer\_lstm(units = 32) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model.rnn.2000 %>% compile(  
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = c("acc")  
)  
  
history.rnn.2000 <- model.rnn.2000 %>% fit(  
 x\_train\_2000, y\_train\_2000,  
 epochs = 20,  
 batch\_size = 32,  
 validation\_data = list(x\_val\_2000, y\_val\_2000)  
)  
  
plot(history.rnn.2000)

## `geom\_smooth()` using formula 'y ~ x'



library(dplyr)  
library( tidyr)  
  
ModeL\_Type <- c("Embedded Layer", "PT Layer", "PT LSTM Layer")  
  
  
  
"100" <- c(mean(history.100$metrics$val\_acc),  
 mean(history.pt.100$metrics$val\_acc),   
 mean(history.rnn.100$metrics$val\_acc))  
  
"500" <- c(mean(history.500$metrics$val\_acc),  
 mean(history.pt.500$metrics$val\_acc),   
 mean(history.rnn.500$metrics$val\_acc))  
  
"1000" <- c(mean(history.1000$metrics$val\_acc),  
 mean(history.pt.1000$metrics$val\_acc),   
 mean(history.rnn.1000$metrics$val\_acc))  
  
"1500" <- c(mean(history.1500$metrics$val\_acc),  
 mean(history.pt.1500$metrics$val\_acc),   
 mean(history.rnn.1500$metrics$val\_acc))  
  
"2000" <- c(mean(history.2000$metrics$val\_acc),  
 mean(history.pt.2000$metrics$val\_acc),   
 mean(history.rnn.2000$metrics$val\_acc))  
  
AVG\_Val\_Table <- data\_frame(ModeL\_Type, `100`, `500`, `1000`, `1500`, `2000`)

## Warning: `data\_frame()` is deprecated, use `tibble()`.  
## This warning is displayed once per session.

AVG\_Val\_Table

## Warning: `...` is not empty.  
##   
## We detected these problematic arguments:  
## \* `needs\_dots`  
##   
## These dots only exist to allow future extensions and should be empty.  
## Did you misspecify an argument?

## # A tibble: 3 x 6  
## ModeL\_Type `100` `500` `1000` `1500` `2000`  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 Embedded Layer 0.505 0.557 0.641 0.696 0.723  
## 2 PT Layer 0.524 0.584 0.605 0.640 0.652  
## 3 PT LSTM Layer 0.531 0.696 0.764 0.779 0.799

AVG\_Table\_Tidy <- gather(AVG\_Val\_Table,"Sample Size", "AVG Validation Accuracy", 2:6)  
  
AVG\_Table\_Tidy$`Sample Size` <- as.numeric(AVG\_Table\_Tidy$`Sample Size`)  
  
AVG\_Table\_Tidy

## Warning: `...` is not empty.  
##   
## We detected these problematic arguments:  
## \* `needs\_dots`  
##   
## These dots only exist to allow future extensions and should be empty.  
## Did you misspecify an argument?

## # A tibble: 15 x 3  
## ModeL\_Type `Sample Size` `AVG Validation Accuracy`  
## <chr> <dbl> <dbl>  
## 1 Embedded Layer 100 0.505  
## 2 PT Layer 100 0.524  
## 3 PT LSTM Layer 100 0.531  
## 4 Embedded Layer 500 0.557  
## 5 PT Layer 500 0.584  
## 6 PT LSTM Layer 500 0.696  
## 7 Embedded Layer 1000 0.641  
## 8 PT Layer 1000 0.605  
## 9 PT LSTM Layer 1000 0.764  
## 10 Embedded Layer 1500 0.696  
## 11 PT Layer 1500 0.640  
## 12 PT LSTM Layer 1500 0.779  
## 13 Embedded Layer 2000 0.723  
## 14 PT Layer 2000 0.652  
## 15 PT LSTM Layer 2000 0.799

#Creating Visual comparison  
  
library(ggplot2)  
  
viz <- ggplot(AVG\_Table\_Tidy, aes(`Sample Size`,`AVG Validation Accuracy`)) +   
 geom\_line(aes(color = ModeL\_Type)) +   
 ggtitle("Validation Accuracy with Varied Training Sample Saizes")  
  
viz

