R Notebook

# Loading the IMDB dataset

library(keras)  
library(tensorflow)  
library(tidyverse)

## -- Attaching packages --------------------------------------------------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1 v purrr 0.3.3  
## v tibble 2.1.3 v dplyr 0.8.3  
## v tidyr 1.0.0 v stringr 1.4.0  
## v readr 1.3.1 v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(cowplot)

##   
## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Note: As of version 1.0.0, cowplot does not change the

## default ggplot2 theme anymore. To recover the previous

## behavior, execute:  
## theme\_set(theme\_cowplot())

## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

imdb <- dataset\_imdb( num\_words = 10000)  
  
c(c( train\_data, train\_lables), c(test\_data, test\_lables)) %<-% imdb

# Encoding the integer sequences into a binary matrix

vectorize\_sequence <- function (sequences, dimension = 10000 ) {  
   
 results <- matrix (0, nrow = length(sequences), ncol = dimension)  
 for ( i in 1: length(sequences))  
 results[i, sequences[[i]]] <- 1  
 results  
}  
  
x\_train <- vectorize\_sequence(train\_data)  
x\_test <- vectorize\_sequence(test\_data)

# Converting labels to numeric

y\_train <- as.numeric(train\_lables)  
y\_test <- as.numeric(test\_lables)

# Setting aside a validation set

val\_indices <- 1:10000  
  
x\_val <- x\_train[val\_indices,]  
partial\_x\_train <- x\_train[-val\_indices,]  
  
  
y\_val <- y\_train[val\_indices]  
partial\_y\_train <- y\_train[-val\_indices]

# building hypothesis sapce based on 3 hiddend layers with 32 as input nodes ans “mse” as loss function

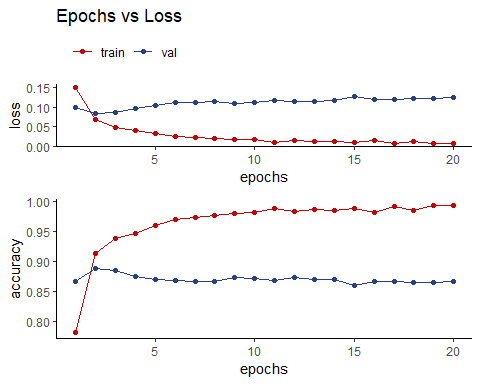
model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 32, activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dense(units = 32, activation = "tanh" ) %>%   
 layer\_dense(units = 32, activation = "tanh" ) %>%  
 layer\_dense(units = 1, activation = "sigmoid")

# Compiling the model

model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )

# Training your model

history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val,y\_val)  
)



**Comments** based on above polt we see model overfitting on training data and not performing well on validation data hecne we’ll tune further paramater of the model

# building hypothesis sapce based on 3 hiddend layers with “64” as input nodes ans “mse” as loss function

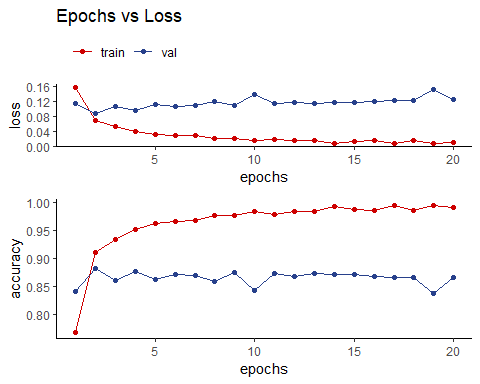
model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 64, activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dense(units = 64, activation = "tanh" ) %>%   
 layer\_dense(units = 64, activation = "tanh" ) %>%  
 layer\_dense(units = 1, activation = "sigmoid")

# Compiling the model

model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )

# Training your model

history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val,y\_val)  
)



**Comments** based on above polt we observe model overfitting on training data and not performing well on validation data hecne we’ll tune further paramater of the model here we’ll add regularization to hidden layer and see if model is performing better on validation

# building hypothesis sapce based on 3 hiddend layers with “32” as input nodes ,“mse” as loss function and “regularizer\_l2(0.001)”

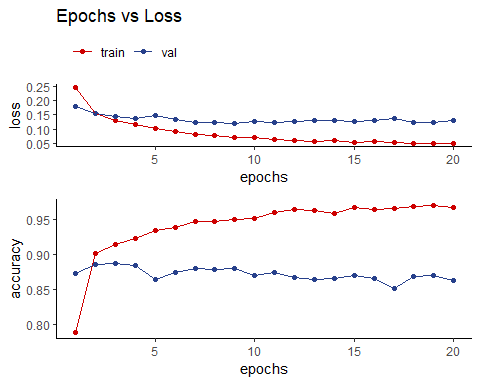
model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l2(0.001) ,activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l2(0.001), activation = "tanh" ) %>%   
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l2(0.001), activation = "tanh" ) %>%  
 layer\_dense(units = 1, activation = "sigmoid")

## Compiling the model

model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )

## Training your model

history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val,y\_val)  
)



**Comments** based on above polt we observe model overfitting on training data and not performing well on validation data hecne we’ll tune further paramater of the model here we’ll add regularization to hidden layer and see if model is performing better on validation

# building hypothesis sapce based on 3 hiddend layers with “32” as input nodes ,“mse” as loss function and “regularizer\_l1(0.001)”

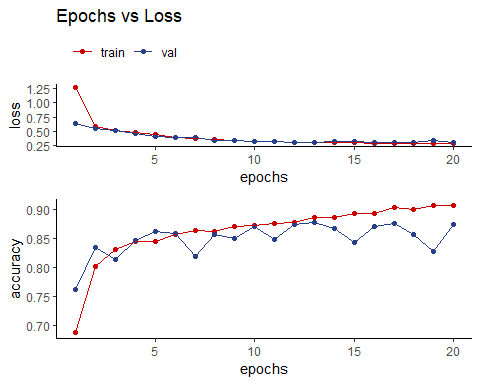
model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001) ,activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%   
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%  
 layer\_dense(units = 1, activation = "sigmoid")

## Compiling the model

model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )

## Training your model

history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val,y\_val)  
)



**Comments** based on above polt we observe model perfomring well on validation data which can be improved by adding dropout to hidden layer

# building hypothesis sapce based on 3 hiddend layers with “32” as input nodes ,“mse” as loss function “regularizer\_l(0.001)” and adding dropout layer with value .50

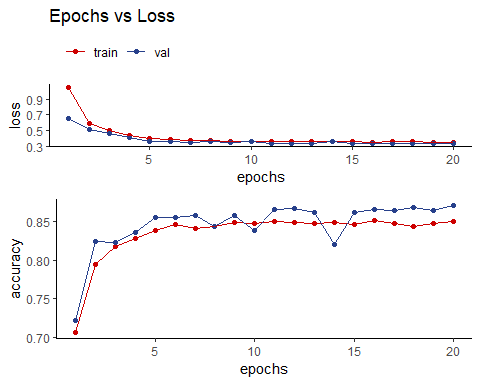
model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001) ,activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%   
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")

## Compiling the model

model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )

## Training your model

history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 250,  
 validation\_data = list(x\_val,y\_val)  
)



**Comments** based on above polt we observe model is performing better on validation set hence moving forward to test on “test” set to find output

# Retraining a model from scratch

model\_r <- keras\_model\_sequential() %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001) ,activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%   
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 32,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
  
model\_r %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )  
  
model\_r %>% fit(  
 x\_train,  
 y\_train,  
 epochs = 10,  
 batch\_size = 250)  
  
result <- model\_r %>% evaluate(x\_test,y\_test)  
result

## $loss  
## [1] 0.3445823  
##   
## $accuracy  
## [1] 0.8616

**COmments** on test set show accuracy of 84%

# building hypothesis sapce based on 3 hiddend layers with “64”

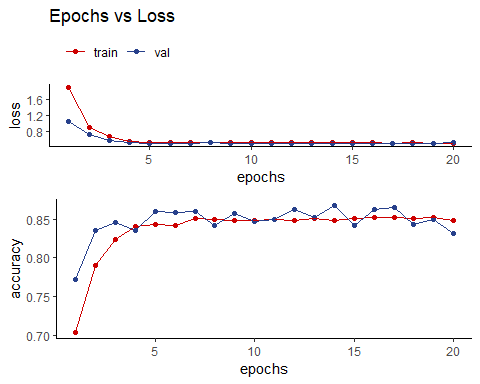
model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 64,kernel\_regularizer = regularizer\_l1(0.001) ,activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 64,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%   
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 64,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")

## Compiling the model

model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )

## Training your model

history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 250,  
 validation\_data = list(x\_val,y\_val)  
)



**Comments** based on above polt we observe model performs is better than “32” and is stable

## Retraining a model from scratch

model\_r <- keras\_model\_sequential() %>%  
 layer\_dense(units = 64,kernel\_regularizer = regularizer\_l1(0.001) ,activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 64,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%   
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 64,kernel\_regularizer = regularizer\_l1(0.001), activation = "tanh" ) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
  
model\_r %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
 )  
  
model\_r %>% fit(  
 x\_train,  
 y\_train,  
 epochs = 9,  
 batch\_size = 250)  
  
result <- model\_r %>% evaluate(x\_test,y\_test)  
result

## $loss  
## [1] 0.5035153  
##   
## $accuracy  
## [1] 0.865

**Comments** based on above model we see model with units = 64 is performing better than units = 32 with improved accuracy of 86% however loss is also increased from 0.3523318 to 0.4975586