Neural Networks Model Hypertuning

Srushti Padade

February 2, 2020

The library Keras is used fo the to load the IMDB Dataset in the workspace. The top most occuring words are considered for our deep learning model.

library(keras)  
  
imdb <- dataset\_imdb(num\_words = 10000)  
c(c(train\_data, train\_labels), c(test\_data, test\_labels)) %<-% imdb  
  
#The dataset is vectorized here by Create an all-zero matrix of shape (len(sequences), dimension) and Sets specific indices of results[i] to 1s.  
  
vectorize\_sequences <- function(sequences, dimension = 10000) {  
 results <- matrix(0, nrow = length(sequences), ncol = dimension)  
 for (i in 1:length(sequences))  
 results[i, sequences[[i]]] <- 1  
 results  
}  
  
#Vectorized the Train and Test data.  
x\_train <- vectorize\_sequences(train\_data)  
x\_test <- vectorize\_sequences(test\_data)  
str(x\_train[1,])

## num [1:10000] 1 1 0 1 1 1 1 1 1 0 ...

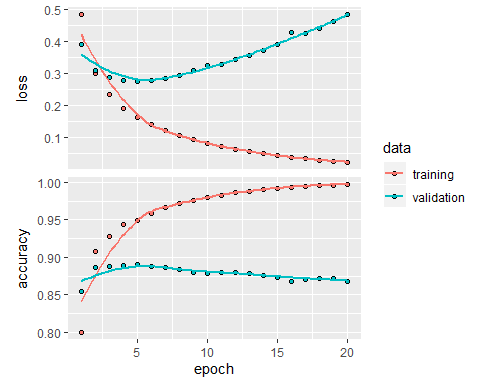
y\_train <- as.numeric(train\_labels)  
y\_test <- as.numeric(test\_labels)  
  
#From the Train data 40 % of data is allocated from Validation data.  
set.seed(123)  
val\_indices <- 1:10000  
  
x\_val <- x\_train[val\_indices,]  
partial\_x\_train <- x\_train[-val\_indices,]  
  
y\_val <- y\_train[val\_indices]  
partial\_y\_train <- y\_train[-val\_indices]

##Layers Hypertuning

#Model 2:

Model Layers - 2

model\_Layer\_1 <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu",input\_shape = c(10000)) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_Layer\_1 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_Layer\_1 %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



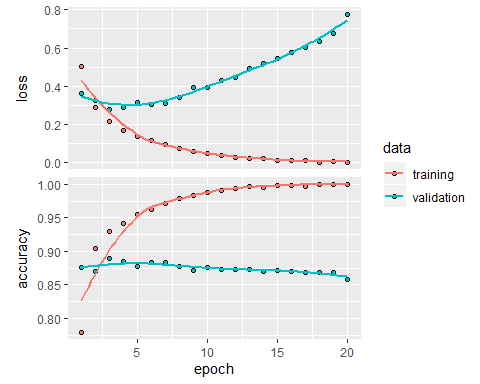
model\_Layer\_1 %>% fit(x\_train, y\_train, epochs = 5, batch\_size = 512)  
Result\_model\_Layer\_1 <- model\_Layer\_1 %>% evaluate(x\_test, y\_test)  
Result\_model\_Layer\_1

## $loss  
## [1] 0.4526824  
##   
## $accuracy  
## [1] 0.86248

#Model 2:

Model Layers - 2

model\_Layer\_2 <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu",input\_shape = c(10000)) %>%  
 layer\_dense(units = 16, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_Layer\_2 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_Layer\_2 %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



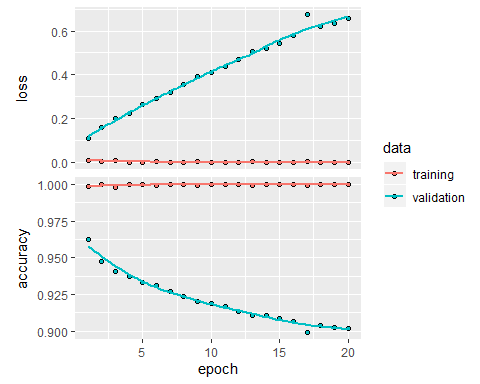
model\_Layer\_2 %>% fit(x\_train, y\_train, epochs = 5, batch\_size = 512)  
Result\_model\_Layer\_2 <- model\_Layer\_2 %>% evaluate(x\_test, y\_test)  
Result\_model\_Layer\_2

## $loss  
## [1] 0.5512349  
##   
## $accuracy  
## [1] 0.86024

#Model 3:

Model Layers - 3

model\_Layer\_3 <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu",input\_shape = c(10000)) %>%  
 layer\_dense(units = 16, activation = "relu")%>%  
 layer\_dense(units = 16, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_Layer\_3 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_Layer\_2 %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



model\_Layer\_3 %>% fit(x\_train, y\_train, epochs = 1, batch\_size = 512)  
Result\_model\_Layer\_3 <- model\_Layer\_3 %>% evaluate(x\_test, y\_test)  
Result\_model\_Layer\_3

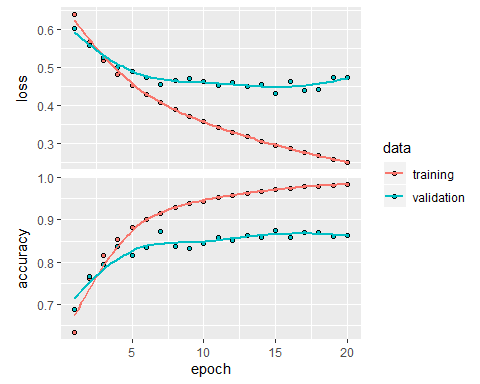
## $loss  
## [1] 0.3431484  
##   
## $accuracy  
## [1] 0.87996

## Units Hypertuning

#Model 4:

Model Units - 4

model\_unit\_4 <- keras\_model\_sequential() %>%   
 layer\_dense(units = 4, activation = "relu",input\_shape = c(10000)) %>%  
 layer\_dense(units = 4, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_unit\_4 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_unit\_4 %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



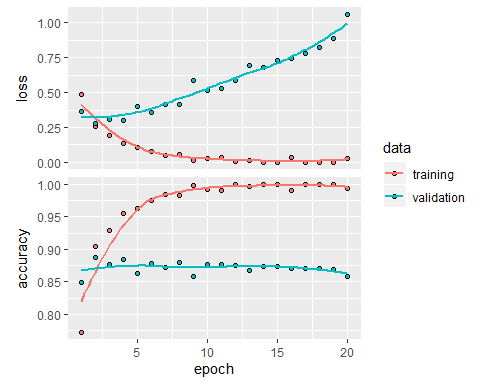
model\_unit\_4 %>% fit(x\_train, y\_train, epochs = 8, batch\_size = 512)  
Result\_model\_unit\_4 <- model\_unit\_4 %>% evaluate(x\_test, y\_test)  
Result\_model\_unit\_4

## $loss  
## [1] 0.4651925  
##   
## $accuracy  
## [1] 0.86

#Model 5:

Model Units - 64

model\_unit\_64 <- keras\_model\_sequential() %>%   
 layer\_dense(units = 64, activation = "relu",input\_shape = c(10000)) %>%  
 layer\_dense(units = 64, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_unit\_64 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_unit\_64 %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



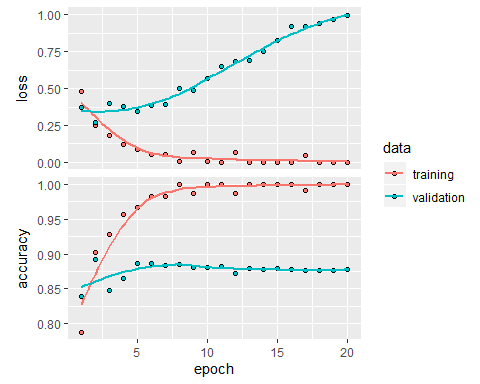
model\_unit\_64 %>% fit(x\_train, y\_train, epochs = 3, batch\_size = 512)  
Result\_model\_unit\_64 <- model\_unit\_64 %>% evaluate(x\_test, y\_test)  
Result\_model\_unit\_64

## $loss  
## [1] 0.500592  
##   
## $accuracy  
## [1] 0.86072

#Model 6:

Model Units - 128

model\_unit\_128 <- keras\_model\_sequential() %>%   
 layer\_dense(units = 128, activation = "relu",input\_shape = c(10000)) %>%  
 layer\_dense(units = 128, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_unit\_128 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_unit\_128 %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



model\_unit\_128 %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
Result\_model\_unit\_128 <- model\_unit\_128 %>% evaluate(x\_test, y\_test)  
Result\_model\_unit\_128

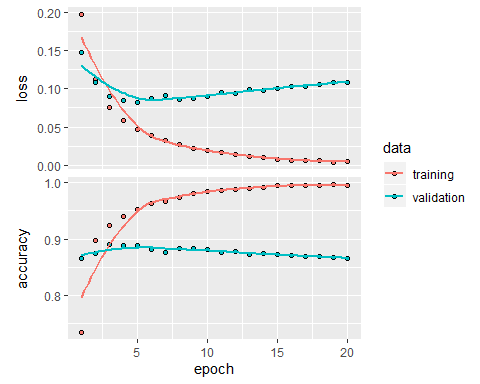
## $loss  
## [1] 0.457048  
##   
## $accuracy  
## [1] 0.86756

## Loss Function

#Model 7:

Model Loss - MSE

model\_mse <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu",input\_shape = c(10000)) %>%  
 layer\_dense(units = 16, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_mse %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history <- model\_mse %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



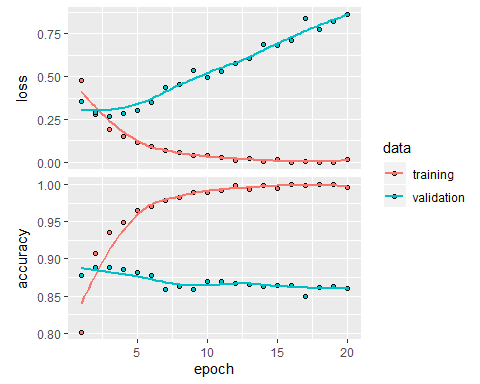
model\_mse %>% fit(x\_train, y\_train, epochs = 4, batch\_size = 512)  
Result\_model\_mse <- model\_mse %>% evaluate(x\_test, y\_test)  
Result\_model\_mse

## $loss  
## [1] 0.1217064  
##   
## $accuracy  
## [1] 0.85388

## Activation Function

Model Activation function - tanh

model\_tanh <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "tanh",input\_shape = c(10000)) %>%  
 layer\_dense(units = 16, activation = "tanh")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_tanh %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_tanh %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



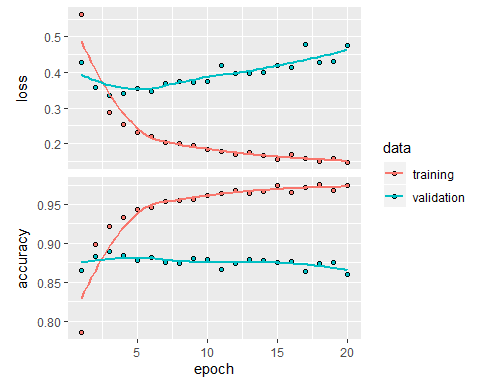
model\_tanh %>% fit(x\_train, y\_train, epochs = 3, batch\_size = 512)  
Result\_model\_tanh <- model\_tanh %>% evaluate(x\_test, y\_test)  
Result\_model\_tanh

## $loss  
## [1] 0.4694537  
##   
## $accuracy  
## [1] 0.85816

## Regularisation

Model Regularizer - L2

model\_reg\_2 <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu", kernel\_regularizer = regularizer\_l2(l = 0.001), input\_shape = c(10000)) %>%  
 layer\_dense(units = 16, activation = "relu", kernel\_regularizer = regularizer\_l2(l = 0.001))%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_reg\_2 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_reg\_2 %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



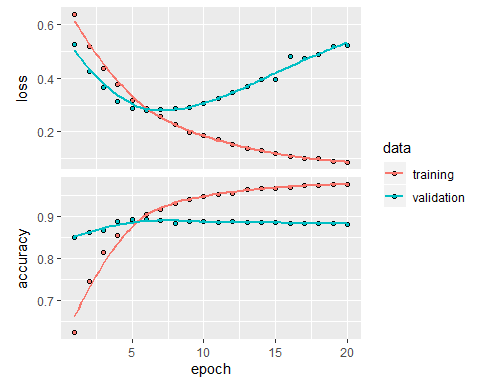
model\_reg\_2 %>% fit(x\_train, y\_train, epochs = 4, batch\_size = 512)  
Result\_model\_reg\_2 <- model\_reg\_2 %>% evaluate(x\_test, y\_test)  
Result\_model\_reg\_2

## $loss  
## [1] 0.4608817  
##   
## $accuracy  
## [1] 0.85824

## Dropout

Model Droupout - 0.5

model\_dropout <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu",input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 16, activation = "relu")%>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_dropout %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- model\_dropout %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



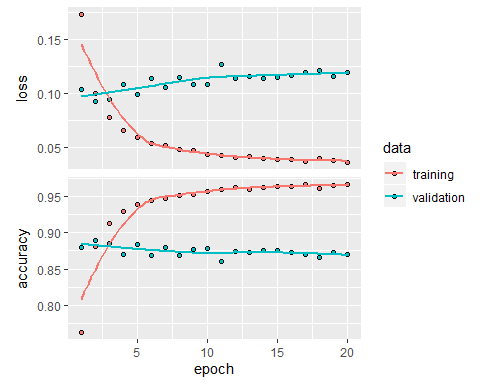
model\_dropout %>% fit(x\_train, y\_train, epochs = 8, batch\_size = 512)  
Result\_model\_dropout <- model\_dropout %>% evaluate(x\_test, y\_test)  
Result\_model\_dropout

## $loss  
## [1] 0.4961424  
##   
## $accuracy  
## [1] 0.8704

## Final Model with Hypertunned Parameters.

model\_Final <- keras\_model\_sequential() %>%   
 layer\_dense(units = 32, activation = "relu",kernel\_regularizer = regularizer\_l2(l = 0.0001), input\_shape = c(10000)) %>%   
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 32, activation = "relu",kernel\_regularizer = regularizer\_l2(l = 0.0001)) %>%   
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 16, activation = "relu"",kernel\_regularizer = regularizer\_l2(l = 0.0001)) %>%

layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model\_Final %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history <- model\_Final %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val))  
  
plot(history)



model\_Final %>% fit(x\_train, y\_train, epochs = 3, batch\_size = 512)  
Result\_model\_Final <- model\_Final %>% evaluate(x\_test, y\_test)  
Result\_model\_Final

## $loss  
## [1] 0.08867288  
##   
## $accuracy  
## [1] 0.8814