Hypertuning

Rithin

2/4/2020

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\_labels), c(test\_data, test\_labels)) %<-% imdb

## preparing the data

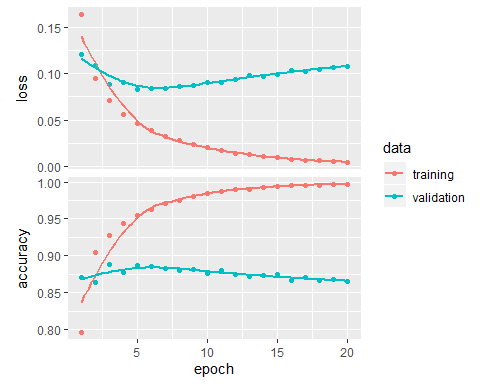
vectorize\_sequences <- function(sequences, dimension = 10000) {  
 # Create an all-zero matrix of shape (len(sequences), dimension)  
 results <- matrix(0, nrow = length(sequences), ncol = dimension)  
 for (i in 1:length(sequences))  
 # Sets specific indices of results[i] to 1s  
 results[i, sequences[[i]]] <- 1  
 results  
}  
  
# Our vectorized training data  
x\_train <- vectorize\_sequences(train\_data)  
# Our vectorized test data  
x\_test <- vectorize\_sequences(test\_data)  
# Our vectorized labels  
y\_train <- as.numeric(train\_labels)  
y\_test <- as.numeric(test\_labels)  
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]

## hidden layer

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

## List of 2  
## $ params :List of 7  
## ..$ batch\_size : int 512  
## ..$ epochs : int 20  
## ..$ steps : num 30  
## ..$ samples : int 15000  
## ..$ verbose : int 0  
## ..$ do\_validation: logi TRUE  
## ..$ metrics : chr [1:4] "loss" "accuracy" "val\_loss" "val\_accuracy"  
## $ metrics:List of 4  
## ..$ loss : num [1:20] 0.1638 0.0946 0.0708 0.0556 0.0462 ...  
## ..$ accuracy : num [1:20] 0.796 0.905 0.928 0.944 0.954 ...  
## ..$ val\_loss : num [1:20] 0.1211 0.1086 0.0884 0.0906 0.0828 ...  
## ..$ val\_accuracy: num [1:20] 0.87 0.863 0.889 0.878 0.887 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)

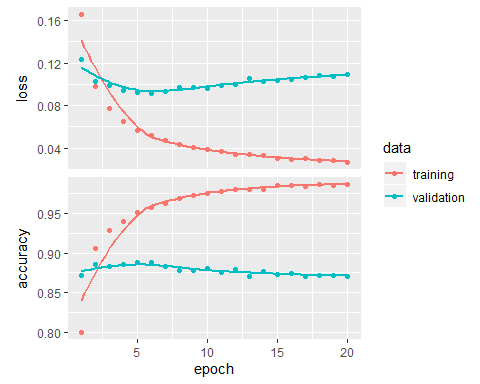
 This approach overfits let us try another with regularization and drop out

## hideen layer with regularization

model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, kernel\_regularizer = regularizer\_l2(.0001),activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
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]  
  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
str(history)

## List of 2  
## $ params :List of 7  
## ..$ batch\_size : int 512  
## ..$ epochs : int 20  
## ..$ steps : num 30  
## ..$ samples : int 15000  
## ..$ verbose : int 0  
## ..$ do\_validation: logi TRUE  
## ..$ metrics : chr [1:4] "loss" "accuracy" "val\_loss" "val\_accuracy"  
## $ metrics:List of 4  
## ..$ loss : num [1:20] 0.1655 0.0982 0.0772 0.0653 0.0564 ...  
## ..$ accuracy : num [1:20] 0.8 0.906 0.928 0.94 0.951 ...  
## ..$ val\_loss : num [1:20] 0.1231 0.1024 0.0986 0.0939 0.0928 ...  
## ..$ val\_accuracy: num [1:20] 0.871 0.885 0.883 0.886 0.888 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)

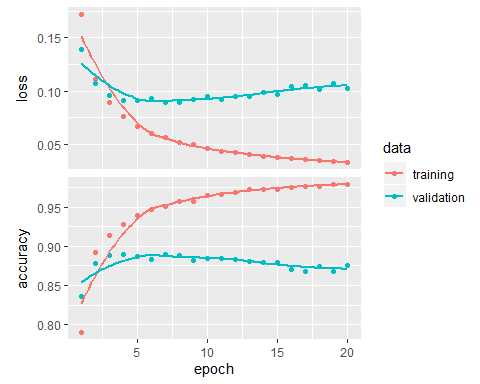
 This network also appears to be overfit after one epoch

## hidden layer with regularization and drop out

model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, kernel\_regularizer = regularizer\_l2(.0001),activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
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]  
  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
str(history)

## List of 2  
## $ params :List of 7  
## ..$ batch\_size : int 512  
## ..$ epochs : int 20  
## ..$ steps : num 30  
## ..$ samples : int 15000  
## ..$ verbose : int 0  
## ..$ do\_validation: logi TRUE  
## ..$ metrics : chr [1:4] "loss" "accuracy" "val\_loss" "val\_accuracy"  
## $ metrics:List of 4  
## ..$ loss : num [1:20] 0.1716 0.1112 0.0893 0.076 0.067 ...  
## ..$ accuracy : num [1:20] 0.791 0.892 0.913 0.928 0.939 ...  
## ..$ val\_loss : num [1:20] 0.1387 0.1076 0.0956 0.0915 0.0912 ...  
## ..$ val\_accuracy: num [1:20] 0.836 0.878 0.888 0.89 0.887 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)

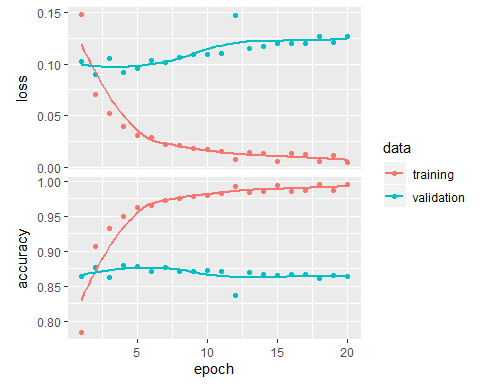
 This model overfits after one epoch, let us try by improvimg accuracy by adding more hiddden layers

## 3 hidden layers with out regularization and drop out

model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 64, activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dense(units = 32, activation = "tanh") %>%  
 layer\_dense(units = 16, activation = "tanh") %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
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]  
  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
str(history)

## List of 2  
## $ params :List of 7  
## ..$ batch\_size : int 512  
## ..$ epochs : int 20  
## ..$ steps : num 30  
## ..$ samples : int 15000  
## ..$ verbose : int 0  
## ..$ do\_validation: logi TRUE  
## ..$ metrics : chr [1:4] "loss" "accuracy" "val\_loss" "val\_accuracy"  
## $ metrics:List of 4  
## ..$ loss : num [1:20] 0.148 0.0705 0.0523 0.0395 0.031 ...  
## ..$ accuracy : num [1:20] 0.785 0.906 0.932 0.949 0.962 ...  
## ..$ val\_loss : num [1:20] 0.1024 0.0898 0.1052 0.0922 0.0962 ...  
## ..$ val\_accuracy: num [1:20] 0.864 0.877 0.863 0.88 0.879 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)

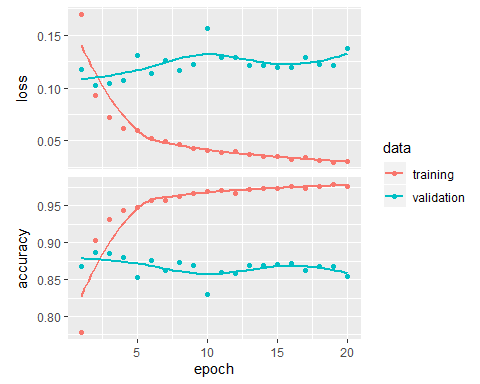
 This layer also overfits aftyer one epoch let us improve accuracy by adding regularization

## 3 hidden layers with regularization

model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 64, kernel\_regularizer = regularizer\_l2(.0001), activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dense(units = 32, kernel\_regularizer = regularizer\_l2(.0001), activation = "tanh") %>%  
 layer\_dense(units = 16, kernel\_regularizer = regularizer\_l2(.0001), activation = "tanh") %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
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]  
  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
str(history)

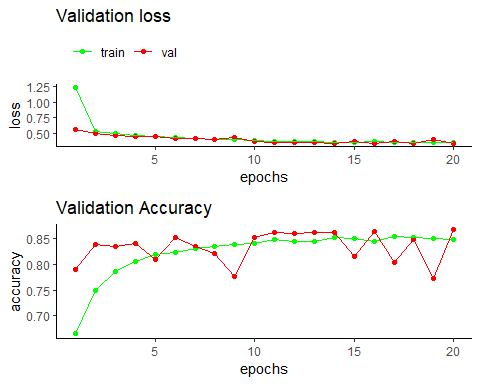
## List of 2  
## $ params :List of 7  
## ..$ batch\_size : int 512  
## ..$ epochs : int 20  
## ..$ steps : num 30  
## ..$ samples : int 15000  
## ..$ verbose : int 0  
## ..$ do\_validation: logi TRUE  
## ..$ metrics : chr [1:4] "loss" "accuracy" "val\_loss" "val\_accuracy"  
## $ metrics:List of 4  
## ..$ loss : num [1:20] 0.1695 0.0934 0.0724 0.0618 0.0597 ...  
## ..$ accuracy : num [1:20] 0.779 0.903 0.931 0.943 0.947 ...  
## ..$ val\_loss : num [1:20] 0.118 0.102 0.104 0.107 0.131 ...  
## ..$ val\_accuracy: num [1:20] 0.867 0.886 0.885 0.88 0.853 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)

 This layer also overfits after one epoch. lets us try with same hidden layers with regularization and drop out

## 3 hidden layers with regularization and drop out

model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 32, kernel\_regularizer = regularizer\_l1(.001), activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 16, kernel\_regularizer = regularizer\_l1(.001), activation = "tanh") %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 16, kernel\_regularizer = regularizer\_l1(.001), activation = "tanh") %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history <- model %>% fit(  
 partial\_x\_train,  
 partial\_y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
hist1 <- as.data.frame(history$metrics)  
names(hist1) <- c("train-loss","train-accuracy","val\_loss","val\_accuracy")  
hist1 <- hist1 %>% mutate(epochs = 1:n()) %>% gather("split","values",-epochs) %>% separate(split,c("split","metric")) %>% spread(metric,values)  
  
 g1<- ggplot(hist1,aes(x=epochs,y=loss,color=split)) + geom\_point() + geom\_line() + theme\_classic() + ggtitle("Validation loss") + theme(legend.position = "top",legend.justification = "left",legend.title = element\_blank()) +scale\_color\_manual(values = c("green","red"))  
 g2<- ggplot(hist1,aes(x=epochs,y=accuracy,color=split)) + geom\_point(show.legend = F) + geom\_line(show.legend = F) + theme\_classic() + ggtitle("Validation Accuracy") + theme(legend.position = "top",legend.justification = "left",legend.title = element\_blank()) + scale\_color\_manual(values = c("green","red"))   
plot\_grid(g1,g2,nrow=2)

 This network overfits after two epoch, Here validation loss is 10% and accuracy is 88%. This model with 3 hidden layers with regularization and drop out is the best model built. we train a new model and evaluate on the test data

model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 64, kernel\_regularizer = regularizer\_l1(.0001), activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 32, kernel\_regularizer = regularizer\_l1(.0001), activation = "tanh") %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 16, kernel\_regularizer = regularizer\_l1(.0001), activation = "tanh") %>%  
 layer\_dropout(rate = 0.5) %>%  
 layer\_dense(units = 1, activation = "sigmoid")  
  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
model %>% fit(x\_train, y\_train, epochs = 8, batch\_size = 512)  
results <- model %>% evaluate(x\_test, y\_test)

## prediction on new data

results

## $loss  
## [1] 0.2009202  
##   
## $accuracy  
## [1] 0.87716

model %>% predict(x\_test[1:10,])

## [,1]  
## [1,] 0.091577321  
## [2,] 0.996323824  
## [3,] 0.989128232  
## [4,] 0.958117962  
## [5,] 0.986166954  
## [6,] 0.904063702  
## [7,] 0.995429039  
## [8,] 0.009762853  
## [9,] 0.984100580  
## [10,] 0.987435341