IMDB dataset-Hypertuning

#Loading IMDB dataset from keras.

library(keras)

## Warning: package 'keras' was built under R version 3.6.2

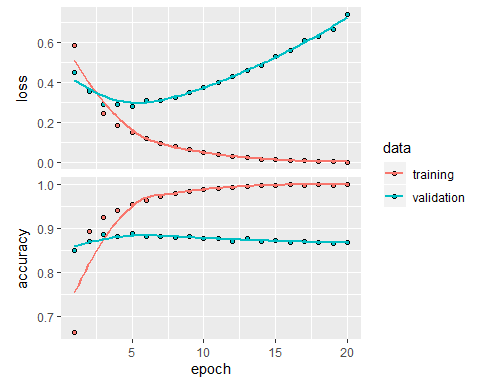
imdb <- dataset\_imdb(num\_words = 10000)  
c(c(train\_data, train\_labels), c(test\_data, test\_labels)) %<-% imdb  
  
vectorize\_sequences <- function(sequences, dimension = 10000) {  
 # Create an all-zero matrix of shape (len(sequences), dimension)  
 results <- matrix(0, nrow = length(sequences), 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  
train <- vectorize\_sequences(train\_data)  
# Our vectorized test data  
test <- vectorize\_sequences(test\_data)  
  
# Our vectorized labels  
train\_label <- as.numeric(train\_labels)  
test\_label <- as.numeric(test\_labels)

#Divide training set into train and validation set for future validating purposes.

val\_indices <- 1:10000  
  
x\_val <- train[val\_indices,]  
x\_train <-train[-val\_indices,]  
  
y\_val <- train\_label[val\_indices]  
y\_train <- train\_label[-val\_indices]

#Model building (neural network) with initial model setting/parameters as layers=3,hidden units=16,activation functions as “relu”,Loss function “binary cross entropy” and optimizer as rmsprop.

initial\_model <- 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")  
  
initial\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history <- initial\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history)



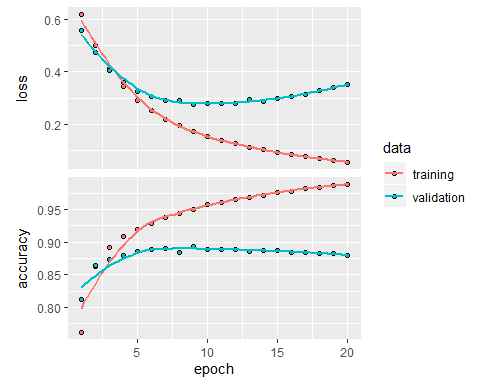
initial\_model%>% fit(train, train\_label, epochs = 4, batch\_size = 512)  
results\_initial <- initial\_model %>% evaluate(test,test\_label)  
results\_initial

## $loss  
## [1] 0.492779  
##   
## $accuracy  
## [1] 0.86248

#Different hidden units like 4,64,128

#Now lets start with hidden units as 4 and all other parameters are same as initial model

units4\_model <- 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")  
  
units4\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
history\_4units <- units4\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_4units)

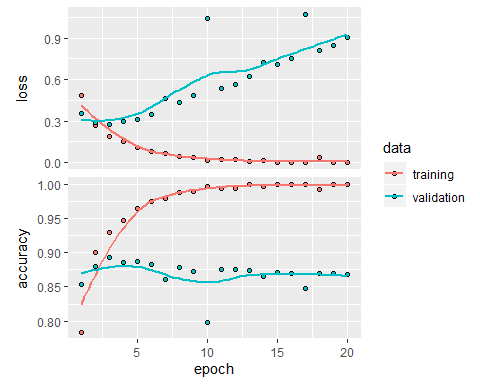


units4\_model%>% fit(train, train\_label, epochs = 4, batch\_size = 512)  
results\_units4 <- units4\_model %>% evaluate(test,test\_label)  
results\_units4

## $loss  
## [1] 0.3708104  
##   
## $accuracy  
## [1] 0.87008

#Hidden units as 64 and remaining parameters same as initial model

units64\_model <- 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")  
  
units64\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
history\_64units <- units64\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_64units)

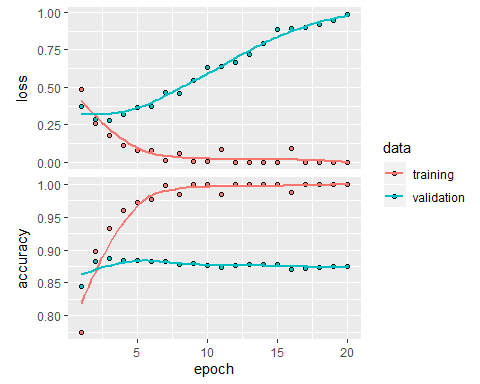


units64\_model%>% fit(train, train\_label, epochs = 2, batch\_size = 512)  
results\_units64 <- units64\_model %>% evaluate(test,test\_label)  
results\_units64

## $loss  
## [1] 0.5353018  
##   
## $accuracy  
## [1] 0.84264

#Increase more hidden units like to 128.

units128\_model <- 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")  
  
units128\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
history\_128units <- units128\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_128units)

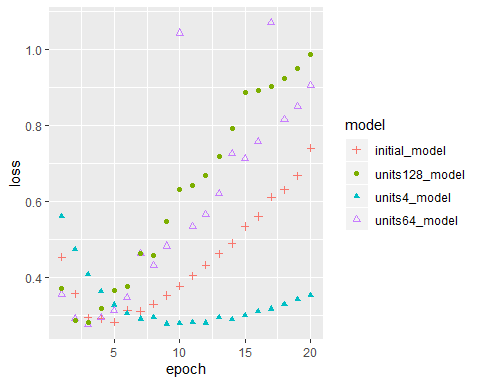


units128\_model%>% fit(train, train\_label, epochs = 2, batch\_size = 512)  
results\_units128 <- units128\_model %>% evaluate(test,test\_label)  
results\_units128

## $loss  
## [1] 0.4464421  
##   
## $accuracy  
## [1] 0.869

#Plotting different hidden units losses

library(ggplot2)  
library(tidyr)  
plot\_val\_losses <- function(losses) {  
 loss\_names <- names(losses)  
 losses <- as.data.frame(losses)  
 losses$epoch <- seq\_len(nrow(losses))  
 losses %>%   
 gather(model, loss, loss\_names[[1]], loss\_names[[2]],loss\_names[[3]],loss\_names[[4]]) %>%   
 ggplot(aes(x = epoch, y = loss, colour = model)) +  
 geom\_point(aes(shape = model))+  
 scale\_shape\_manual(values=c(3,16,17,2))  
}  
plot\_val\_losses(losses = list(  
 initial\_model = history$metrics$val\_loss,  
 units4\_model = history\_4units$metrics$val\_loss,  
 units64\_model = history\_64units$metrics$val\_loss,  
 units128\_model = history\_128units$metrics$val\_loss))

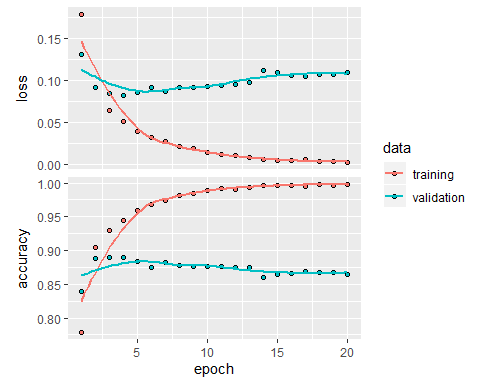


# smaller hidden units of 4 model starts overfitting much later than initial model and when hidden units are increased to 64 or 128 the models overfits early like with 128 units loss is increased in second epoch itself.

#mse loss function

#Now changing loss function from binary cross entropy to “mse” and all other parameters as same.

mse\_model <- 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")  
  
mse\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "mse",  
 metrics = c("accuracy")  
)  
  
history\_mse <- mse\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_mse)



mse\_model%>% fit(train, train\_label, epochs = 2, batch\_size = 512)  
results\_mse <- mse\_model %>% evaluate(test,test\_label)  
results\_mse

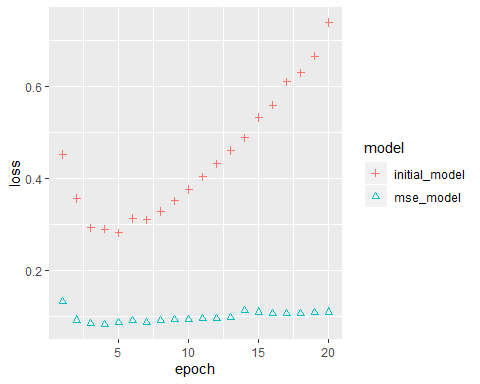
## $loss  
## [1] 0.1128262  
##   
## $accuracy  
## [1] 0.86108

#Using “mse” as loss function for IMDB dataset, loss value is low when compared to binary cross entropy. MSE is also good measure for data of form normal distribution. Let’s examine with an example If actual and predicted labels are same, then both loss functions give error as “0”. Actual label: 1 Predicted label: 1 MSE: (1 - 1) ² = 0 Cross-entropy: -(1 \* log (1) + 0 \* log (0)) = 0

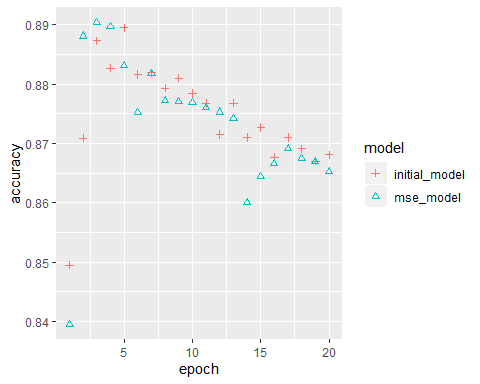
Consider different scenario like Actual label: 1 Predicted label: 0 MSE: (1 - 1) ² = 1 Cross-entropy: -(1 \* log (0) + 0 \* log (1)) = tends to infinity Here “mse” loss function error is less compared to cross entropy loss function. Cross entropy strongly penalizes the misclassifications.

#Plots for loss and accuracy for initial model and mse model.

plot\_val\_losses <- function(losses) {  
 loss\_names <- names(losses)  
 losses <- as.data.frame(losses)  
 losses$epoch <- seq\_len(nrow(losses))  
 losses %>%   
 gather(model, loss, loss\_names[[1]], loss\_names[[2]]) %>%   
 ggplot(aes(x = epoch, y = loss, colour = model)) +  
 geom\_point(aes(shape = model))+  
 scale\_shape\_manual(values=c(3,2))  
}  
plot\_val\_losses(losses = list(  
 initial\_model = history$metrics$val\_loss,  
 mse\_model = history\_mse$metrics$val\_loss  
 ))



plot\_val\_accuracy <- function(accuracy) {  
 accuracy\_names <- names(accuracy)  
 accuracy <- as.data.frame(accuracy)  
 accuracy$epoch <- seq\_len(nrow(accuracy))  
 accuracy %>%   
 gather(model, accuracy, accuracy\_names[[1]], accuracy\_names[[2]]) %>%   
 ggplot(aes(x = epoch, y = accuracy, colour = model)) +  
 geom\_point(aes(shape = model))+  
 scale\_shape\_manual(values=c(3,2))  
}  
plot\_val\_accuracy(accuracy = list(  
 initial\_model = history$metrics$val\_accuracy,  
 mse\_model = history\_mse$metrics$val\_accuracy  
 ))

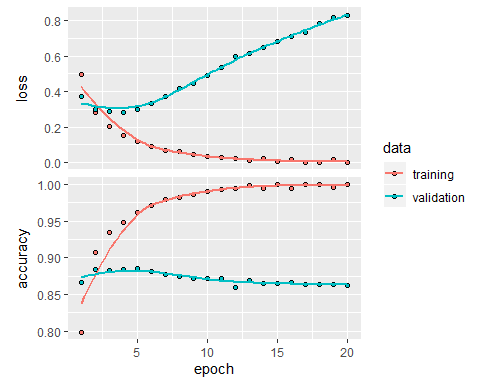


#In above graphs we can clearly notice that loss is less and accuracy is also better for mse as loss function model.

#tanh activation function

#Using tanh instead of ReLu activation function

tanh\_model <- 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")  
  
tanh\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history\_tanh <- tanh\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_tanh)

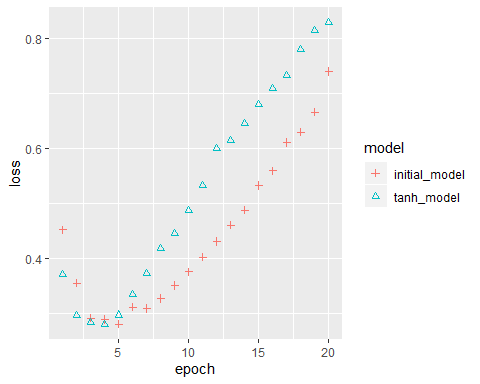


tanh\_model%>% fit(train, train\_label, epochs = 2, batch\_size = 512)  
results\_tanh <- tanh\_model %>% evaluate(test,test\_label)  
results\_tanh

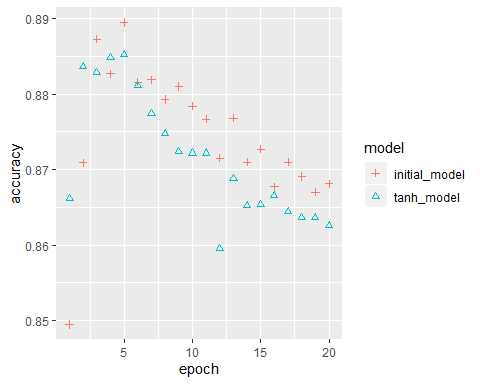
## $loss  
## [1] 0.4718881  
##   
## $accuracy  
## [1] 0.85412

#PLotting loss and accuracy

plot\_val\_losses(losses = list(  
 initial\_model = history$metrics$val\_loss,  
 tanh\_model = history\_tanh$metrics$val\_loss  
 ))

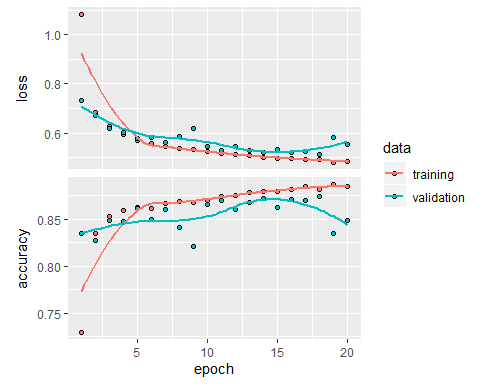


plot\_val\_accuracy(accuracy = list(  
 initial\_model = history$metrics$val\_accuracy,  
 tanh\_model = history\_tanh$metrics$val\_accuracy  
 ))

 #The accuracy of tanh activation function for the model is low and also degrades performance due to vanishing gradient problem.

#L-1 regularization #L-1 norm regularization is cost added is proportional to the absolute values of weight coefficients.

L1\_model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16,kernel\_regularizer = regularizer\_l1(0.001), activation = "relu", input\_shape = c(10000)) %>%   
 layer\_dense(units = 16,kernel\_regularizer = regularizer\_l1(0.001), activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
L1\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history\_L1 <- L1\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_L1)

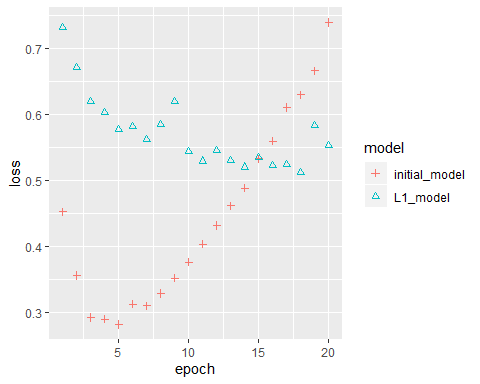


L1\_model%>% fit(train, train\_label, epochs = 4, batch\_size = 512)  
results\_L1 <- L1\_model %>% evaluate(test,test\_label)  
results\_L1

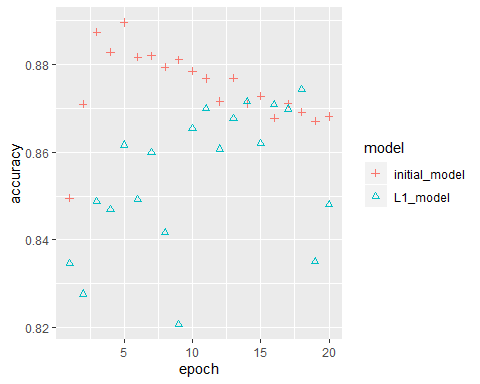
## $loss  
## [1] 0.5019544  
##   
## $accuracy  
## [1] 0.87108

#Plotting loss and accuracy for validation dataset

plot\_val\_losses(losses = list(  
 initial\_model = history$metrics$val\_loss,  
 L1\_model = history\_L1$metrics$val\_loss  
))



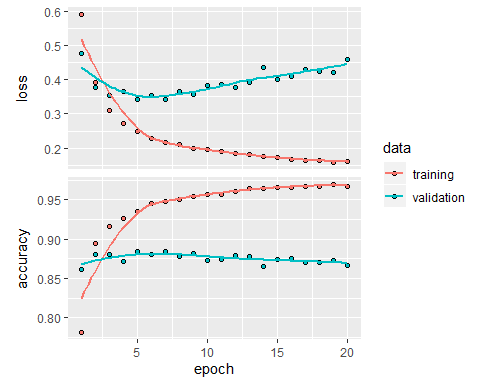
plot\_val\_accuracy(accuracy = list(  
 initial\_model = history$metrics$val\_accuracy,  
 L1\_model = history\_L1$metrics$val\_accuracy  
))



#L-1 model is resistant to overfitting than initial model.Accuracy is almost the same with small differnces in each epoch.

#L-2 regularization #L-2 norm regularization-cost added is proportional to square of the value of weight coefficients.

L2\_model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16,kernel\_regularizer = regularizer\_l2(0.001), activation = "relu", input\_shape = c(10000)) %>%   
 layer\_dense(units = 16,kernel\_regularizer = regularizer\_l2(0.001), activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
L2\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history\_L2 <- L2\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_L2)

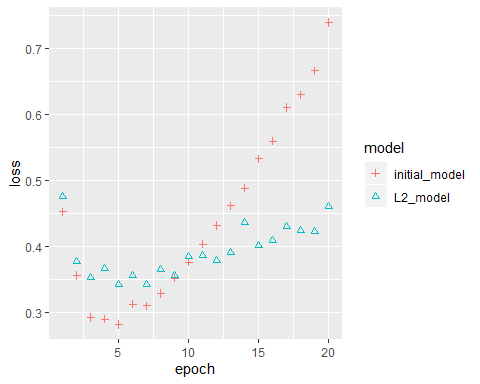


L2\_model%>% fit(train, train\_label, epochs = 4, batch\_size = 512)  
results\_L2 <- L2\_model %>% evaluate(test,test\_label)  
results\_L2

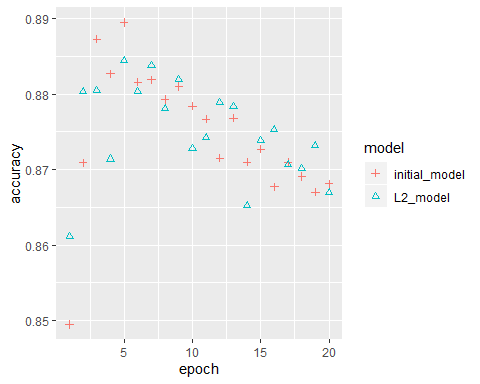
## $loss  
## [1] 0.3955323  
##   
## $accuracy  
## [1] 0.87032

#plot for loss and accuracy

plot\_val\_losses(losses = list(  
 initial\_model = history$metrics$val\_loss,  
 L2\_model = history\_L2$metrics$val\_loss  
))



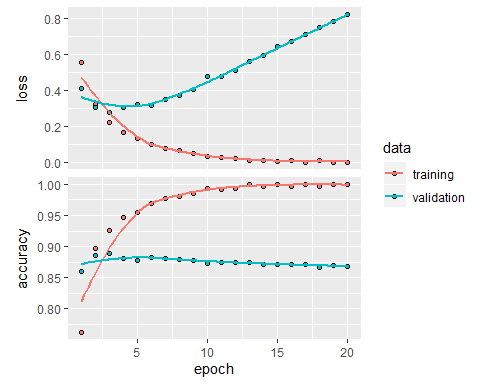
plot\_val\_accuracy(accuracy = list(  
 initial\_model = history$metrics$val\_accuracy,  
 L2\_model = history\_L2$metrics$val\_accuracy  
))

 #The graph for L-2 model shows loss is significantly low and also less overfitting compared to initial model.Coming to the accuracy the L-2 model showing better accuracy.

#References:“Deep Learning with R”-Chollet Allaire. # <https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261> # Lecture notes-MIS64061,MIS64037.

#Differnt number ofHidden Layers # 3 hidden layers

layers3\_model <- 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")  
  
layers3\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history\_layers3 <- layers3\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_layers3)

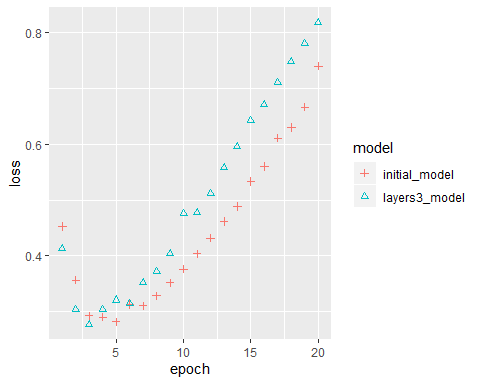


layers3\_model%>% fit(train, train\_label, epochs = 4, batch\_size = 512)  
results\_layers3 <- layers3\_model %>% evaluate(test,test\_label)  
results\_layers3

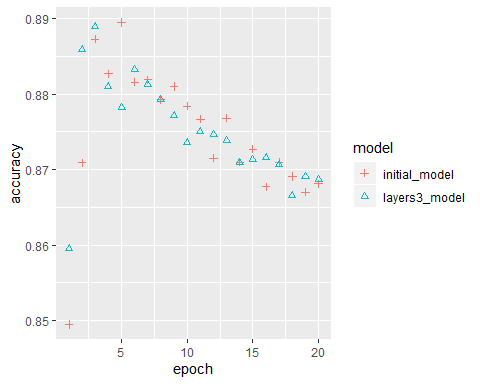
## $loss  
## [1] 0.5122639  
##   
## $accuracy  
## [1] 0.85996

#Plot

plot\_val\_losses(losses = list(  
 initial\_model = history$metrics$val\_loss,  
 layers3\_model = history\_layers3$metrics$val\_loss  
))

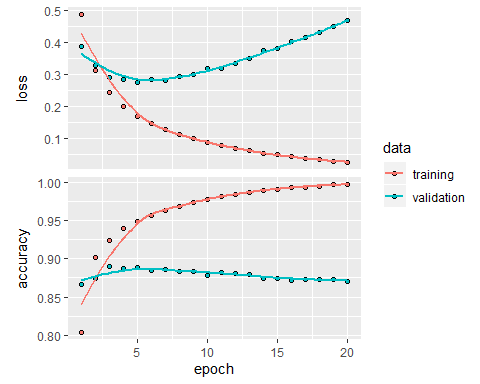


plot\_val\_accuracy(accuracy = list(  
 initial\_model = history$metrics$val\_accuracy,  
 layers3\_model = history\_layers3$metrics$val\_accuracy  
))



#one hidden layer

layers1\_model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu", input\_shape = c(10000)) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
layers1\_model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = c("accuracy")  
)  
  
history\_layers1 <- layers1\_model %>% fit(  
 x\_train, y\_train,  
 epochs = 20,  
 batch\_size = 512,  
 validation\_data = list(x\_val, y\_val)  
)  
  
plot(history\_layers1)

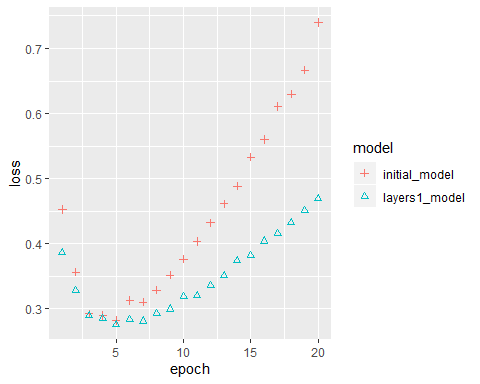


layers1\_model%>% fit(train, train\_label, epochs = 4, batch\_size = 512)  
results\_layers1 <- layers1\_model %>% evaluate(test,test\_label)  
results\_layers1

## $loss  
## [1] 0.4218043  
##   
## $accuracy  
## [1] 0.8608

# Plot for loss and accuarcy

plot\_val\_losses(losses = list(  
 initial\_model = history$metrics$val\_loss,  
 layers1\_model = history\_layers1$metrics$val\_loss  
))



plot\_val\_accuracy(accuracy = list(  
 initial\_model = history$metrics$val\_accuracy,  
 layers1\_model = history\_layers1$metrics$val\_accuracy  
))

