Assignment-1

Group 7

2/3/2020

library(keras)  
imdb <- dataset\_imdb(num\_words = 10000)# top 10,000 most frequently occurring words in the training data  
c(c(train\_data, train\_labels), c(test\_data, test\_labels)) %<-% imdb  
# Preparing 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)  
str(x\_train[1,])

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

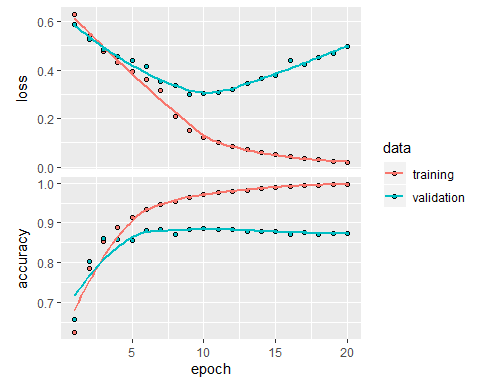
# Our vectorized labels  
y\_train <- as.numeric(train\_labels)  
y\_test <- as.numeric(test\_labels)  
val\_indices <- 1:10000  
x\_val <- x\_train[val\_indices,]  
partial\_x\_train <- x\_train[-val\_indices,]  
y\_val <- y\_train[val\_indices]  
partial\_y\_train <- y\_train[-val\_indices]

# 8 units

# Using layers with 8 hidden units rather than 16 units  
  
model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 8, activation = "relu",input\_shape = c(10000)) %>%   
 layer\_dense(units = 8, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.628 0.537 0.478 0.433 0.395 ...  
## ..$ accuracy : num [1:20] 0.625 0.786 0.853 0.889 0.914 ...  
## ..$ val\_loss : num [1:20] 0.587 0.526 0.487 0.457 0.441 ...  
## ..$ val\_accuracy: num [1:20] 0.656 0.802 0.86 0.858 0.855 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 3, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

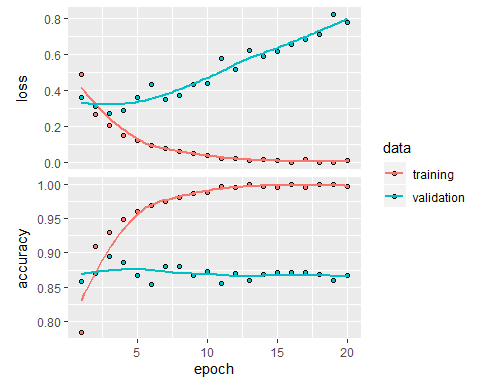
## $loss  
## [1] 0.427411  
##   
## $accuracy  
## [1] 0.86272

# 32 units

# Using layers with 32 hidden units rather than 16 units  
  
model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 32, activation = "relu",input\_shape = c(10000)) %>%   
 layer\_dense(units = 32, activation = "relu")%>%  
 layer\_dense(units = 1, activation = "sigmoid")  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.49 0.269 0.203 0.152 0.121 ...  
## ..$ accuracy : num [1:20] 0.785 0.909 0.929 0.948 0.961 ...  
## ..$ val\_loss : num [1:20] 0.362 0.313 0.271 0.287 0.361 ...  
## ..$ val\_accuracy: num [1:20] 0.859 0.871 0.895 0.887 0.868 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

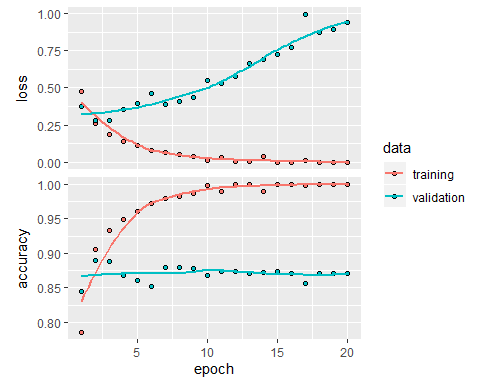
## $loss  
## [1] 0.4122058  
##   
## $accuracy  
## [1] 0.86264

# 64 units

# Using layers with 64 hidden units rather than 16 units  
  
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")  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.479 0.26 0.186 0.141 0.11 ...  
## ..$ accuracy : num [1:20] 0.786 0.906 0.933 0.949 0.961 ...  
## ..$ val\_loss : num [1:20] 0.372 0.278 0.282 0.355 0.398 ...  
## ..$ val\_accuracy: num [1:20] 0.845 0.89 0.888 0.868 0.86 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

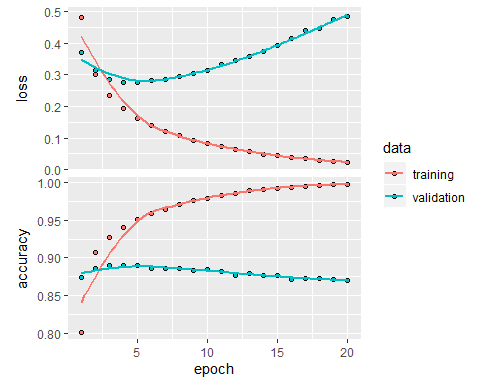
## $loss  
## [1] 0.3922156  
##   
## $accuracy  
## [1] 0.86608

# Single Hidden Layer

# Using one hidden layer  
  
model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 16, activation = "relu",input\_shape = c(10000)) %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.482 0.302 0.233 0.192 0.163 ...  
## ..$ accuracy : num [1:20] 0.802 0.907 0.927 0.94 0.95 ...  
## ..$ val\_loss : num [1:20] 0.371 0.313 0.284 0.276 0.275 ...  
## ..$ val\_accuracy: num [1:20] 0.874 0.886 0.891 0.891 0.89 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

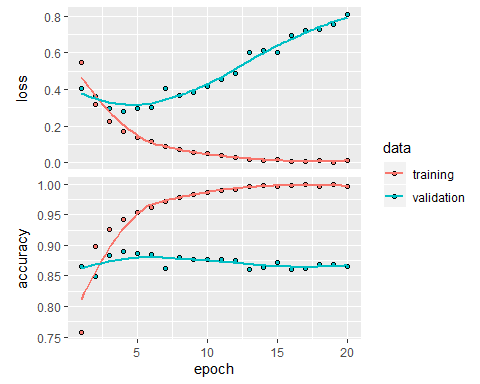
## $loss  
## [1] 0.409004  
##   
## $accuracy  
## [1] 0.8608

# Third Hidden Layer

# Using three hidden layers   
  
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")  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.547 0.316 0.223 0.17 0.138 ...  
## ..$ accuracy : num [1:20] 0.758 0.898 0.927 0.943 0.954 ...  
## ..$ val\_loss : num [1:20] 0.404 0.359 0.294 0.278 0.298 ...  
## ..$ val\_accuracy: num [1:20] 0.866 0.849 0.883 0.89 0.887 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

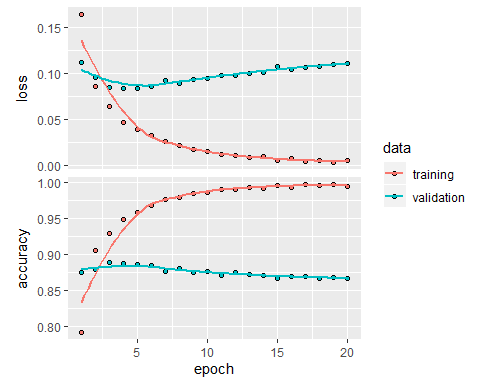
## $loss  
## [1] 0.4610373  
##   
## $accuracy  
## [1] 0.85976

# MSE (Mean Squared Error)

# Using Loss Function "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")  
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] 0.1637 0.086 0.0634 0.0467 0.0384 ...  
## ..$ accuracy : num [1:20] 0.792 0.905 0.929 0.949 0.959 ...  
## ..$ val\_loss : num [1:20] 0.1119 0.0949 0.0848 0.0832 0.0833 ...  
## ..$ val\_accuracy: num [1:20] 0.876 0.879 0.889 0.887 0.886 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

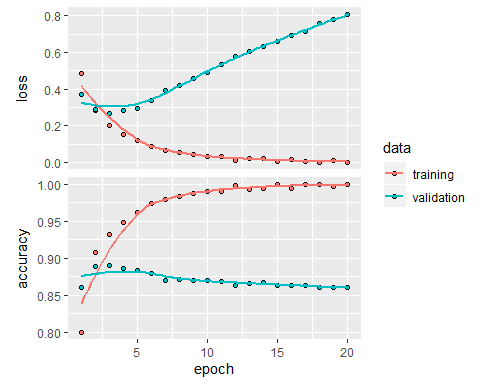
## $loss  
## [1] 0.114676  
##   
## $accuracy  
## [1] 0.85964

# Tanh

# Using "tanh" activation instead of "relu"  
  
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")  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.485 0.283 0.199 0.152 0.118 ...  
## ..$ accuracy : num [1:20] 0.8 0.908 0.933 0.949 0.962 ...  
## ..$ val\_loss : num [1:20] 0.373 0.288 0.27 0.282 0.297 ...  
## ..$ val\_accuracy: num [1:20] 0.861 0.888 0.89 0.886 0.884 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

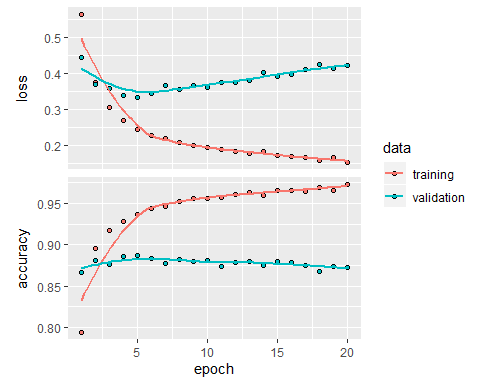
## $loss  
## [1] 0.438916  
##   
## $accuracy  
## [1] 0.85572

# Regularization

# Using Regularization technique for tuning the function by adding an additional penalty term in the error function.  
  
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")  
model %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.563 0.375 0.304 0.268 0.245 ...  
## ..$ accuracy : num [1:20] 0.795 0.896 0.917 0.928 0.937 ...  
## ..$ val\_loss : num [1:20] 0.445 0.37 0.357 0.339 0.332 ...  
## ..$ val\_accuracy: num [1:20] 0.867 0.881 0.876 0.886 0.887 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

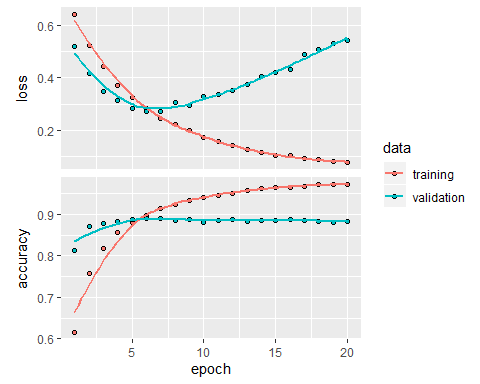
## $loss  
## [1] 0.4149688  
##   
## $accuracy  
## [1] 0.863

# Drop-out

# Using Drop-out technique which refers to dropping out units to prevent overfitting  
  
model <- 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 %>% compile(  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 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] 0.641 0.524 0.442 0.373 0.324 ...  
## ..$ accuracy : num [1:20] 0.616 0.758 0.818 0.858 0.881 ...  
## ..$ val\_loss : num [1:20] 0.521 0.418 0.349 0.314 0.283 ...  
## ..$ val\_accuracy: num [1:20] 0.813 0.871 0.879 0.883 0.889 ...  
## - attr(\*, "class")= chr "keras\_training\_history"

plot(history)

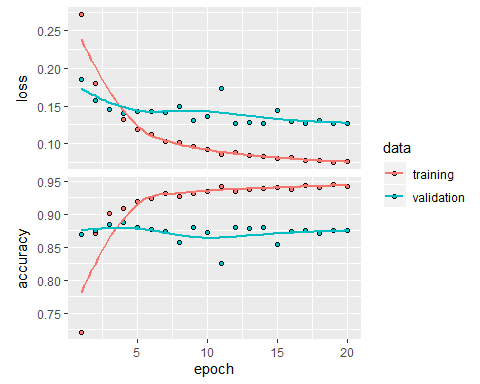


model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

## $loss  
## [1] 0.4123228  
##   
## $accuracy  
## [1] 0.87484

# Final Model Applying the used hypertuning parameters.

model <- keras\_model\_sequential() %>%   
 layer\_dense(units = 32, kernel\_regularizer = regularizer\_l2(0.001), activation = "tanh", input\_shape = c(10000)) %>%  
 layer\_dropout(rate=0.5) %>%  
 layer\_dense(units = 16, activation = "tanh", kernel\_regularizer = regularizer\_l2(0.001)) %>%  
 layer\_dropout(rate=0.5) %>%  
 layer\_dense(units = 32, activation = "tanh",kernel\_regularizer = regularizer\_l2(0.001)) %>%  
 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))  
  
plot(history)



model %>% fit(x\_train, y\_train, epochs = 2, batch\_size = 512)  
model %>% evaluate(x\_test, y\_test)

## $loss  
## [1] 0.1188882  
##   
## $accuracy  
## [1] 0.87736

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

## [,1]  
## [1,] 0.026100904  
## [2,] 0.999923110  
## [3,] 0.906235695  
## [4,] 0.911288917  
## [5,] 0.978154957  
## [6,] 0.981089830  
## [7,] 0.999729872  
## [8,] 0.001732826  
## [9,] 0.995793819  
## [10,] 0.997836769

Here we could find the minimum loss function with good accuracy.