Assignment_01

Code By: Xan Lamoreux

Advanced Machine Learning

Import Keras Library and load imdb dataset

num_words = 10000 (To keep the top 10000 most frequent words)

```
library(keras)
imdb <- dataset_imdb(num_words = 10000)

## Loaded Tensorflow version 2.6.0
```

Get the list of reviews for train and test data and labels

Labels 0 means negative review Labels 1 means positive review

```
train_data <- imdb$train$x
train_labels <- imdb$train$y
test_data <- imdb$test$x
test_labels <- imdb$test$y
```

Check one of data set value and its label

```
str(train_data[[11]])

##  int [1:450] 1 785 189 438 47 110 142 7 6 7475 ...

train_labels[[11]]

## [1] 1
```

Maximum index number

```
max(sapply(train_data, max))

## [1] 9999
```

Decoding Review

```
word_index <- dataset_imdb_word_index()
word_index[11]

## $`hold's`
## [1] 52009
```

```
reverse_word_index <- names(word_index)
reverse_word_index[11]

## [1] "hold's"

names(reverse_word_index) <- word_index

decoded_review <- sapply(train_data[[11]], function(index) {
  word <- if (index >= 3) reverse_word_index[[as.character(index - 3)]]
  if (!is.null(word)) word else "?"
})

cat(decoded_review)

## ? french horror cinema has seen something of a revival over the last coupl
e of years with great films such as inside and ? romance ? on to the scene ?
? the revival just slightly but stands head and shoulders over most modern ho
rror titles and is surely one of the best french horror films ever made ? was
obviously shot on a low budget but this is made up for in far more ways than
one by the originality of the film and this in turn is ? by the excellent wri
ting and acting that ensure the film is a winner the plot focuses on two main
ideas prison and black magic the central character is a man named ? sent to p
rison for fraud he is put in a cell with three others the quietly insane ? bo
dy building ? marcus and his retarded boyfriend daisy after a short while in
the cell together they stumble upon a hiding place in the wall that contains
an old ? after ? part of it they soon realise its magical powers and realise
they may be able to use it to break through the prison walls br br black magi
c is a very interesting topic and i'm actually quite surprised that there are
n't more films based on it as there's so much scope for things to do with it
it's fair to say that ? makes the best of it's ? as despite it's ? the film n
ever actually feels restrained and manages to flow well throughout director e
ric ? provides a great atmosphere for the film the fact that most of it takes
place inside the central prison cell ? that the film feels very claustrophobi
c and this immensely benefits the central idea of the prisoners wanting to us
e magic to break out of the cell it's very easy to get behind them it's often
said that the unknown is the thing that really ? people and this film proves
that as the director ? that we can never really be sure of exactly what is ro
und the corner and this helps to ensure that ? actually does manage to be qui
te frightening the film is memorable for a lot of reasons outside the central
plot the characters are all very interesting in their own way and the fact th
at the book itself almost takes on its own character is very well done anyone
worried that the film won't deliver by the end won't be disappointed either a
s the ending both makes sense and manages to be quite horrifying overall ? is
a truly great horror film and one of the best of the decade highly recommende
d viewing
```

Preparing the Data

Use one-hot-encoding technique to turn list into vectors of 0’s and 1’s

```
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
}

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)
```

Creating 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]
```

Build the network

Create the sequential model

```
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")
```

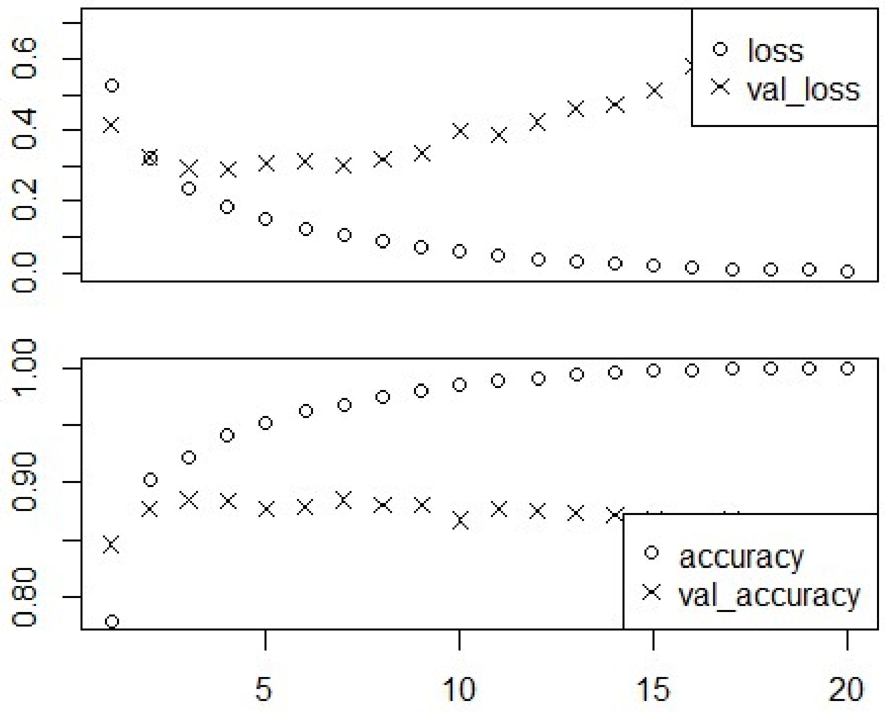
Compiling the model

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

Train the model

```
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 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.528 0.32 0.239 0.186 0.152 ...
##   ..$ accuracy    : num [1:20] 0.78 0.903 0.922 0.941 0.953 ...
##   ..$ val_loss    : num [1:20] 0.417 0.324 0.293 0.29 0.307 ...
##   ..$ val_accuracy: num [1:20] 0.846 0.876 0.885 0.883 0.877 ...
##  - attr(*, "class")= chr "keras_training_history"

plot(history)
```



Creating model from scratch and test it

```
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 = "binary_crossentropy",
  metrics = c("accuracy")
)

model %>% fit(
  x_train,
  y_train,
  epochs = 4,
  batch_size = 512)

results <- model %>% evaluate(x_test, y_test)

results

##      loss  accuracy
## 0.2963269 0.8827200
```

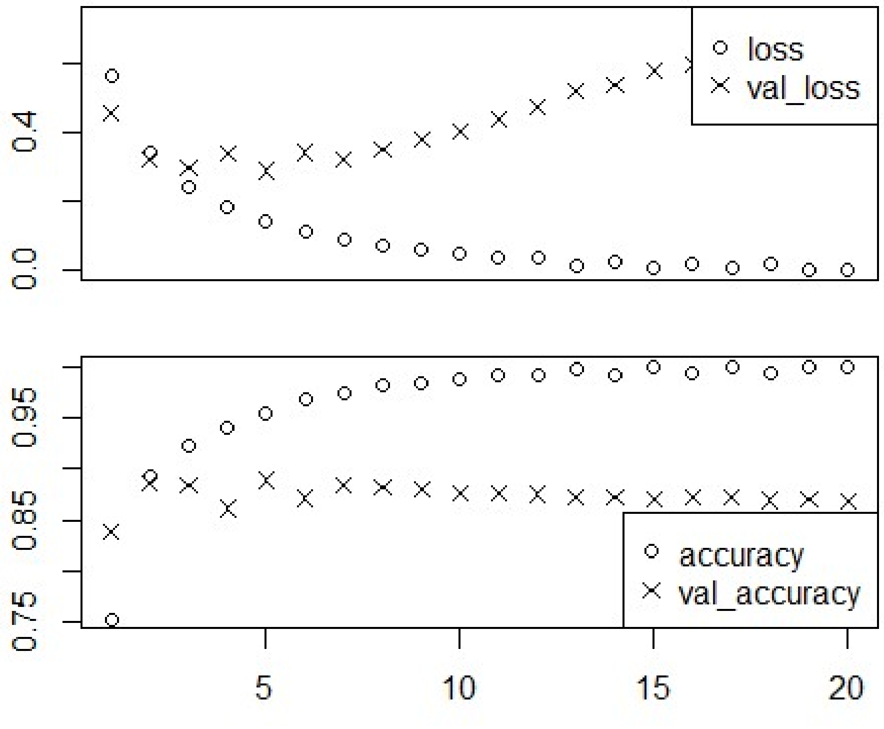
Check predictions on new data set

```
model %>% predict(x_test[1:10,])

##              [,1]
##  [1,] 0.169404119
##  [2,] 0.999827385
##  [3,] 0.896921992
##  [4,] 0.857095599
##  [5,] 0.965436399
##  [6,] 0.877163529
##  [7,] 0.999889851
##  [8,] 0.009294242
##  [9,] 0.973581314
## [10,] 0.994523168
```

Task # 01.

You used two hidden layers. Try using one or three hidden layers, and see how doing so affects validation and test accuracy.

```
# USING 3 HIDDEN LAYERS
model_01 <- 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_01 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

history_01 <- model_01 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)

str(history_01)

## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.568 0.343 0.241 0.181 0.143 ...
##   ..$ accuracy    : num [1:20] 0.753 0.894 0.924 0.941 0.953 ...
##   ..$ val_loss    : num [1:20] 0.456 0.319 0.297 0.34 0.288 ...
##   ..$ val_accuracy: num [1:20] 0.839 0.887 0.885 0.861 0.889 ...
##  - attr(*, "class")= chr "keras_training_history"

plot(history_01)
```

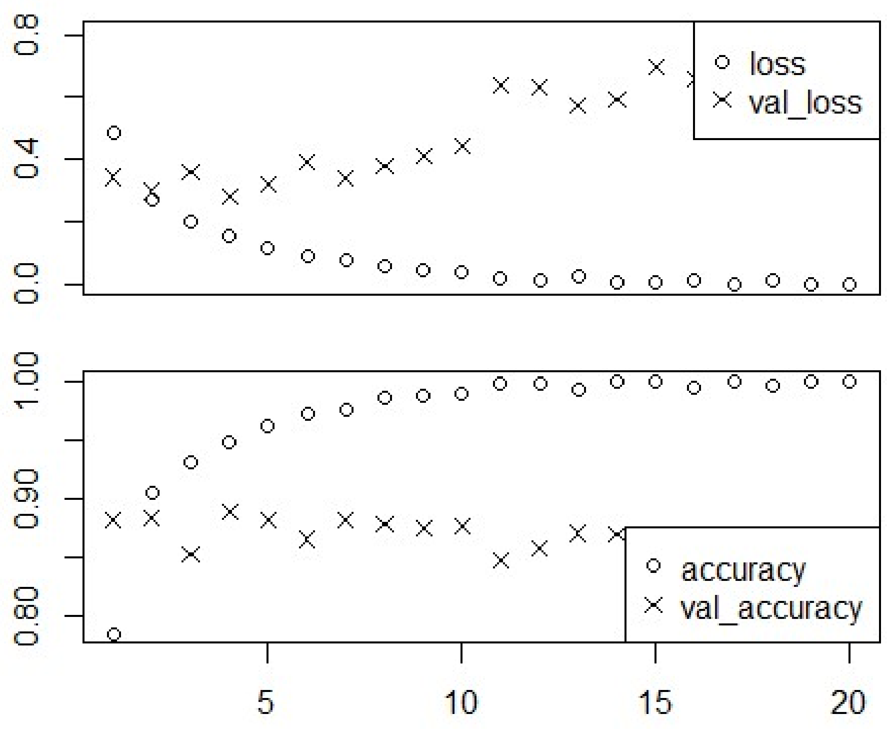


```
results_01 <- model_01 %>% evaluate(x_test, y_test)

results_01

##      loss  accuracy
## 0.8133937 0.8520000

history_01

##
## Final epoch (plot to see history):
##         loss: 0.00155
##     accuracy: 0.9999
##     val_loss: 0.7342
## val_accuracy: 0.8687
```

TASK # 02.

Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.

```
# USING 32 HIDDEN UNITS
model_02 <- 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_02 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

history_02 <- model_02 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)

str(history_02)

## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.484 0.273 0.199 0.155 0.119 ...
##   ..$ accuracy    : num [1:20] 0.785 0.905 0.931 0.948 0.962 ...
##   ..$ val_loss    : num [1:20] 0.343 0.298 0.358 0.28 0.319 ...
##   ..$ val_accuracy: num [1:20] 0.881 0.884 0.853 0.888 0.882 ...
##  - attr(*, "class")= chr "keras_training_history"

plot(history_02)
```

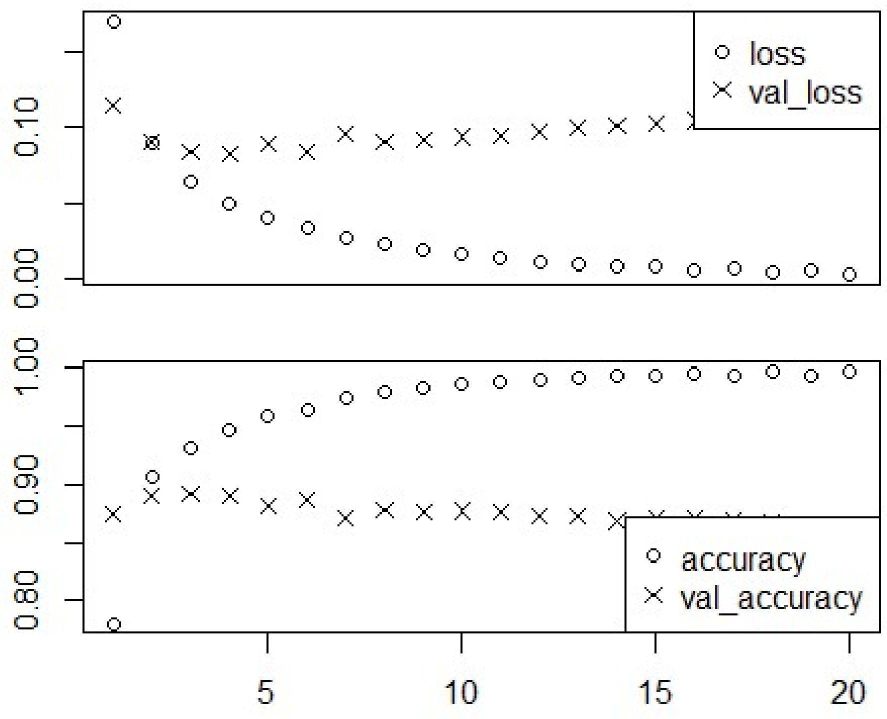


```
results_02 <- model_02 %>% evaluate(x_test, y_test)

results_02

##      loss  accuracy
## 0.8998921 0.8532400

history_02

##
## Final epoch (plot to see history):
##         loss: 0.0007392
##     accuracy: 0.9999
##     val_loss: 0.8088
## val_accuracy: 0.8654
```

TASK # 03.

Try using the mse loss function instead of binary_crossentropy.

```
# USING MSE LOSS FUNCTION
model_03 <- 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_03 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)

history_03 <- model_03 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)

str(history_03)

## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.1694 0.0891 0.0641 0.0497 0.0399 ...
##   ..$ accuracy    : num [1:20] 0.78 0.906 0.931 0.946 0.959 ...
##   ..$ val_loss    : num [1:20] 0.1144 0.0907 0.0838 0.0826 0.0886 ...
##   ..$ val_accuracy: num [1:20] 0.875 0.891 0.892 0.89 0.881 ...
##  - attr(*, "class")= chr "keras_training_history"

plot(history_03)
```

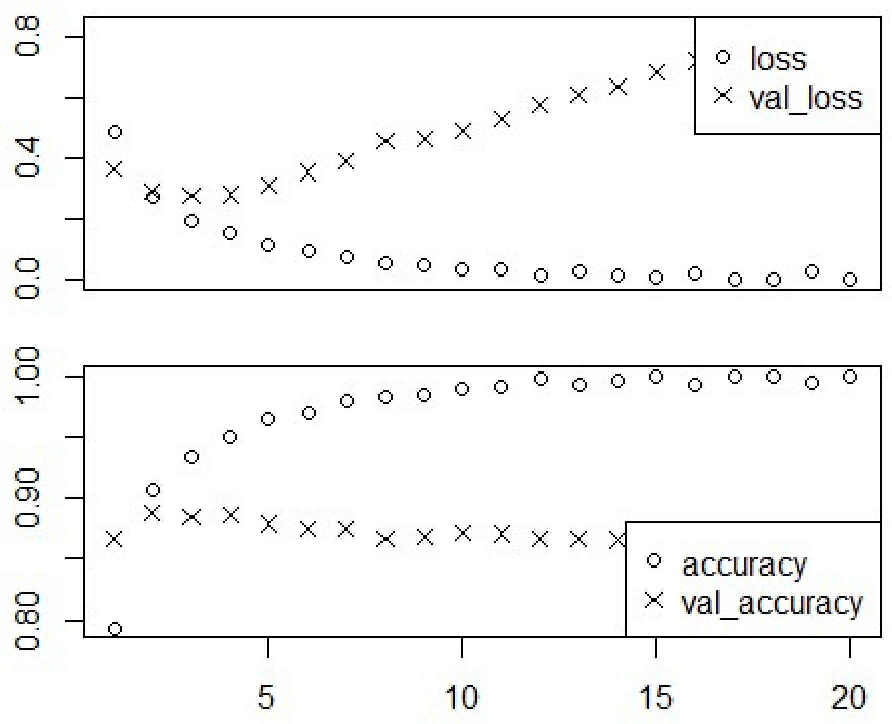


```
results_03 <- model_03 %>% evaluate(x_test, y_test)

results_03

##      loss  accuracy
## 0.1213527 0.8518400

history_03

##
## Final epoch (plot to see history):
##         loss: 0.003199
##     accuracy: 0.9972
##     val_loss: 0.1107
## val_accuracy: 0.8641
```

TASK # 04.

4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.

```
# USING TANH ACTIVATION FUNCTION
model_04 <- 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_04 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

history_04 <- model_04 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)

str(history_04)

## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.486 0.276 0.195 0.149 0.113 ...
##   ..$ accuracy    : num [1:20] 0.794 0.907 0.934 0.95 0.965 ...
##   ..$ val_loss    : num [1:20] 0.364 0.289 0.279 0.28 0.309 ...
##   ..$ val_accuracy: num [1:20] 0.867 0.888 0.885 0.887 0.879 ...
##  - attr(*, "class")= chr "keras_training_history"

plot(history_04)
```

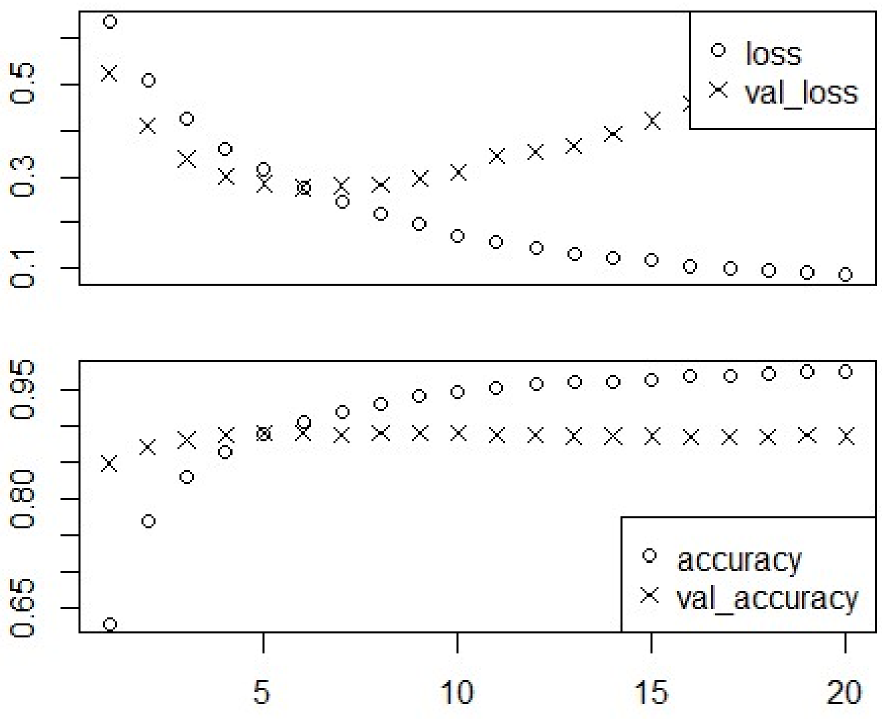


```
results_04 <- model_04 %>% evaluate(x_test, y_test)

results_04

##      loss  accuracy
## 0.9270635 0.8457600

history_04

##
## Final epoch (plot to see history):
##         loss: 0.000481
##     accuracy: 1
##     val_loss: 0.8327
## val_accuracy: 0.86
```

TASK # 05.

5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.

```
# USING DROP OUT

model_05 <- 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_05 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

history_05 <- model_05 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)

str(history_05)

## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.634 0.509 0.423 0.361 0.313 ...
##   ..$ accuracy    : num [1:20] 0.629 0.768 0.829 0.863 0.888 ...
##   ..$ val_loss    : num [1:20] 0.524 0.411 0.338 0.302 0.284 ...
##   ..$ val_accuracy: num [1:20] 0.848 0.869 0.88 0.886 0.889 ...
##  - attr(*, "class")= chr "keras_training_history"

plot(history_05)
```



```
results_05 <- model_05 %>% evaluate(x_test, y_test)

results_05

##      loss  accuracy
## 0.5936748 0.8738800

history_05

##
## Final epoch (plot to see history):
##         loss: 0.08861
##     accuracy: 0.9728
##     val_loss: 0.5524
## val_accuracy: 0.8853
```

SUMMARY

Hypertuning Parameters	Loss	Accuracy	Validation Loss	Validation Accuracy
Using 3 Hidden Layers	0.00155	0.9999	0.7342	0.8687
Using 32 Hidden Units	0.0007392	0.9999	0.8088	0.8654
Using MSE Loss Function	0.003199	0.9972	0.1107	0.8641
Using TANH Activation Function	0.000481	1	0.8327	0.86
Using Drop Out	0.08861	0.9728	0.5524	0.8853

With all hypertuning parameters mentioned in above table, **validation accuracy** was around **86%** but when use **drop out** technique (last one in the above table) then it increased slightly and reach about **86.53%.**