# Deep Learning with R

Chenshu Liu

April 2022

## Contents

# 1 Dense Layer Model

## 1.1 Data & Library

```
# import spreadsheet files
library(readr)
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'hms'
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'tibble'
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'
```

```
# deep learning package
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.1.2
```

```
setwd("~/Documents/Programming/R/Deep Learning with R/Datasets")
data <- read_csv("SimulatedBinaryClassificationDataset.csv",
                 col_names = TRUE)
```

```
## Rows: 50000 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (11): Var1, Var2, Var3, Var4, Var5, Var6, Var7, Var8, Var9, Var10, Target
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

## 1.2 Data Preprocessing

```
# data.frame --> matrix
data  <- as.matrix(data)
# remove the row and col names, leaving only numerical values
dimnames(data) = NULL
mode(data)
```

```
## [1] "numeric"
```

## 1.3 Train Test Split

```r
# train and test split index
set.seed(123)
index <- sample(2,
                nrow(data),
                replace = TRUE,
                prob = c(0.9, 0.1))

# data splitting
x_train <- data[index == 1, 1:10]
x_test <- data[index == 2, 1:10]
y_test_actual <- data[index == 2, 11]

# use teh to_categorical function in keras package for one-hot encoding
y_train <- to_categorical(data[index == 1, 11])
```

```
## Loaded Tensorflow version 2.8.0
```

```r
y_test <- to_categorical(data[index == 2, 11])
```

## 1.4 Modeling

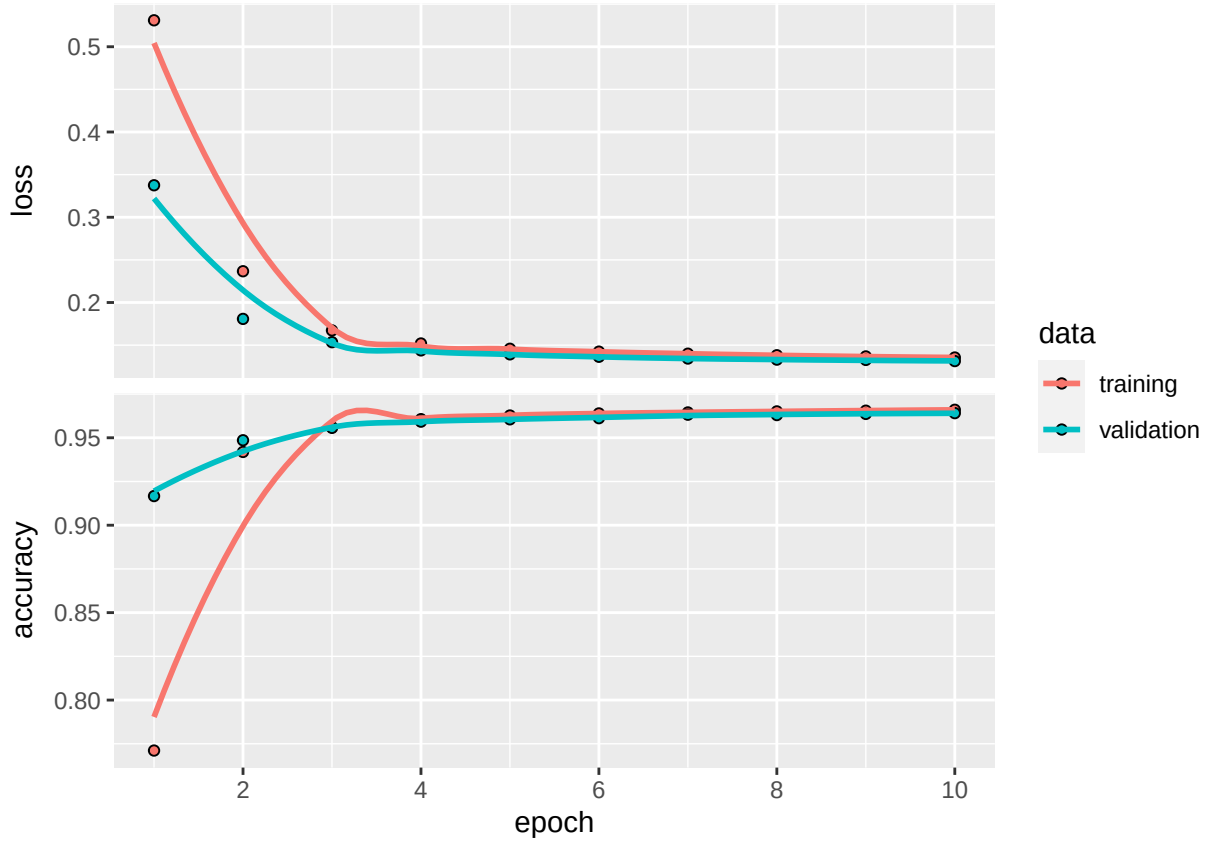
```r
model <- keras_model_sequential() %>%
  # layer_dense means a densely connected layer
  layer_dense(name = "DeepLayer1",
              units = 10, # hyperparameter: the number of nodes
              activation = "relu",
              # the first layer need to have specification about the input dimension
              input_shape = c(10)) %>%
  layer_dense(name = "DeepLayer2",
              units = 10,
              activation = "relu") %>%
  layer_dense(name = "OutputLayer",
              units = 2,
              # softmax function will provide probabilities of the nodes
              activation = "softmax")

summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  DeepLayer1 (Dense)                 (None, 10)                      110
##
##  DeepLayer2 (Dense)                 (None, 10)                      110
##
##  OutputLayer (Dense)                (None, 2)                       22
##
## =========================================================================
## Total params: 242
## Trainable params: 242
```

```
## Non-trainable params: 0
## _________________________________________________________________________
```

```
model %>% compile(
  # another way to calculate loss, besides mean-squared-error
  loss = "categorical_crossentropy",
  # a special way of gradient descent
  optimizer = "adam",
  # measurement of model performance - using accuracy to measure
  metrics = c("accuracy"))

history <- model %>%
  fit(x_train,
      y_train,
      # number of full forward & backward propagation
      # (i.e. run 10 times back and forth of all samples)
      epoch = 10,
      # instead of propagating the whole dataset at one go, use smaller batches
      batch_size = 256,
      # splitting the training set to test itself during training
      validation_split = 0.1,
      verbose = 2)

# plot the training history
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
model %>%
  evaluate(x_test,
           # NOTE: here we are still using the one-hot encoded y_test
           y_test)
```

```
##      loss  accuracy
## 0.1493152 0.9612784
```

```
# form predictions
pred <- model %>%
  predict(x_test) %>%
  k_argmax()
# reference for converting tensor to R data types
# https://torch.mlverse.org/technical/tensors/
pred <- as.array(pred)
table(Predicted = pred,
      # NOTE: for confusion matrix, we are using the original y_test_actual, not encoded
      Actual = y_test_actual)
```

```
##          Actual
## Predicted    0    1
##         0 2419  150
##         1   39 2273
```

# 2 Regularization

## 2.1 Data & Library

```
# Applying regularization to deal with overfitting
library(keras)
library(readr)
library(tidyr)
library(tibble)
library(plotly)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
# specify the number of feature variables for the dataset to be downloaded
num_words <- 5000
imdb <- dataset_imdb(num_words = num_words)

# train test split
c(train_data, train_labels) %<-% imdb$train
c(test_data, test_labels) %<-% imdb$test
```

## 2.2 Data Preprocessing

```
# multi-hot encoding
multi_hot_sequences <- function(sequences, dimension){
  multi_hot <- matrix(0,
                      # the number of samples in the sequences
                      # sequences are stored as lists
                      nrow = length(sequences),
                      ncol = dimension)
  for(i in 1 : length(sequences)){
    # sequences[[i]] extracts the label of the words in the text sample i
    # which ever word is included in that sequence will be assigned 1 at row i
    multi_hot[i, sequences[[i]]] <- 1
```

```
  }
  multi_hot
}

train_data <- multi_hot_sequences(train_data, num_words)
test_data <- multi_hot_sequences(test_data, num_words)
```

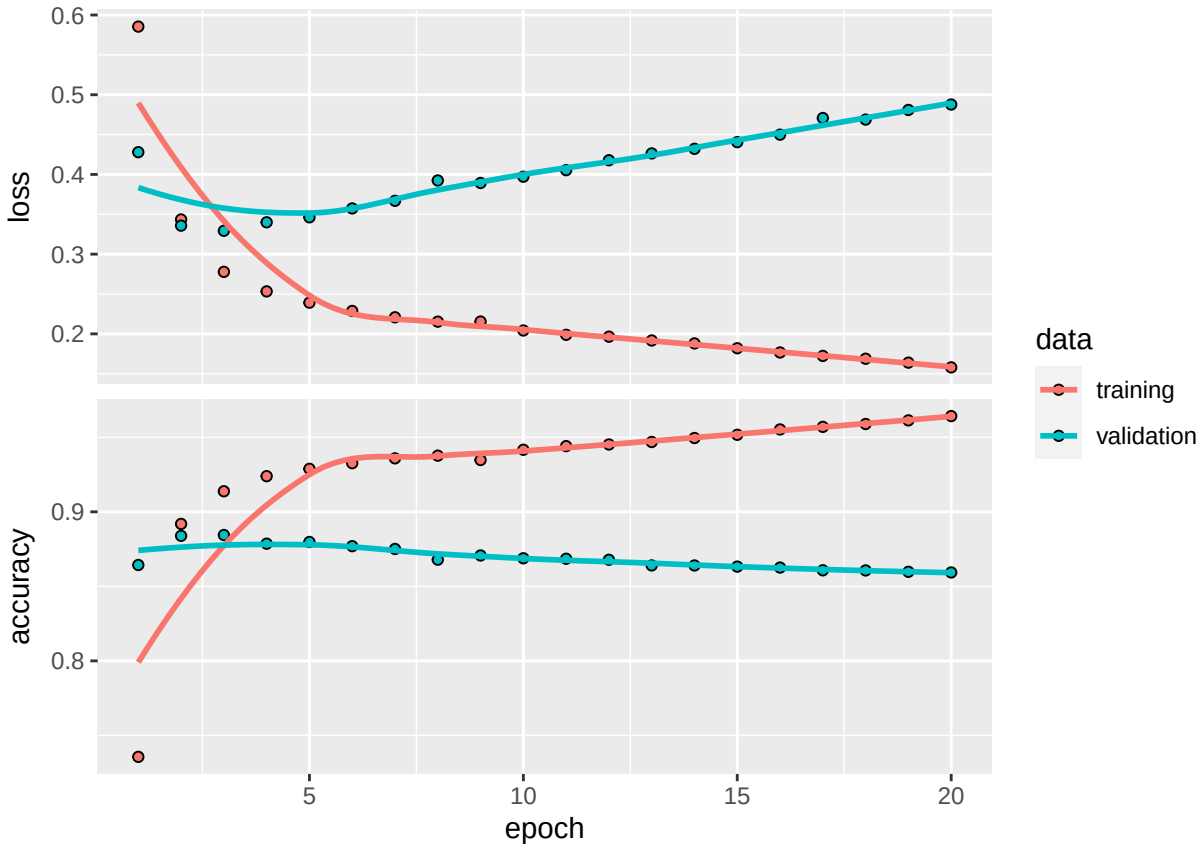
### 2.3 L2 Regularization Model

```
l2_model <-
  keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = num_words,
              # apply regularization in the layer_dense function's argument
              kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 16, activation = "relu",
              kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 1, activation = "sigmoid")

l2_model %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = list("accuracy")
)

l2_model %>% summary()
```

```
## Model: "sequential_1"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_2 (Dense)                    (None, 16)                      80016
##
##  dense_1 (Dense)                    (None, 16)                      272
##
##  dense (Dense)                      (None, 1)                       17
##
## =========================================================================
## Total params: 80,305
## Trainable params: 80,305
## Non-trainable params: 0
## _________________________________________________________________________
```

```
l2_history <- l2_model %>% fit(
  train_data,
  train_labels,
  epoch = 20,
  batch_size = 512,
  validation_data = list(test_data, test_labels),
  verbose = 2
)

plot(l2_history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




## 2.4 Dropout Regularization Model

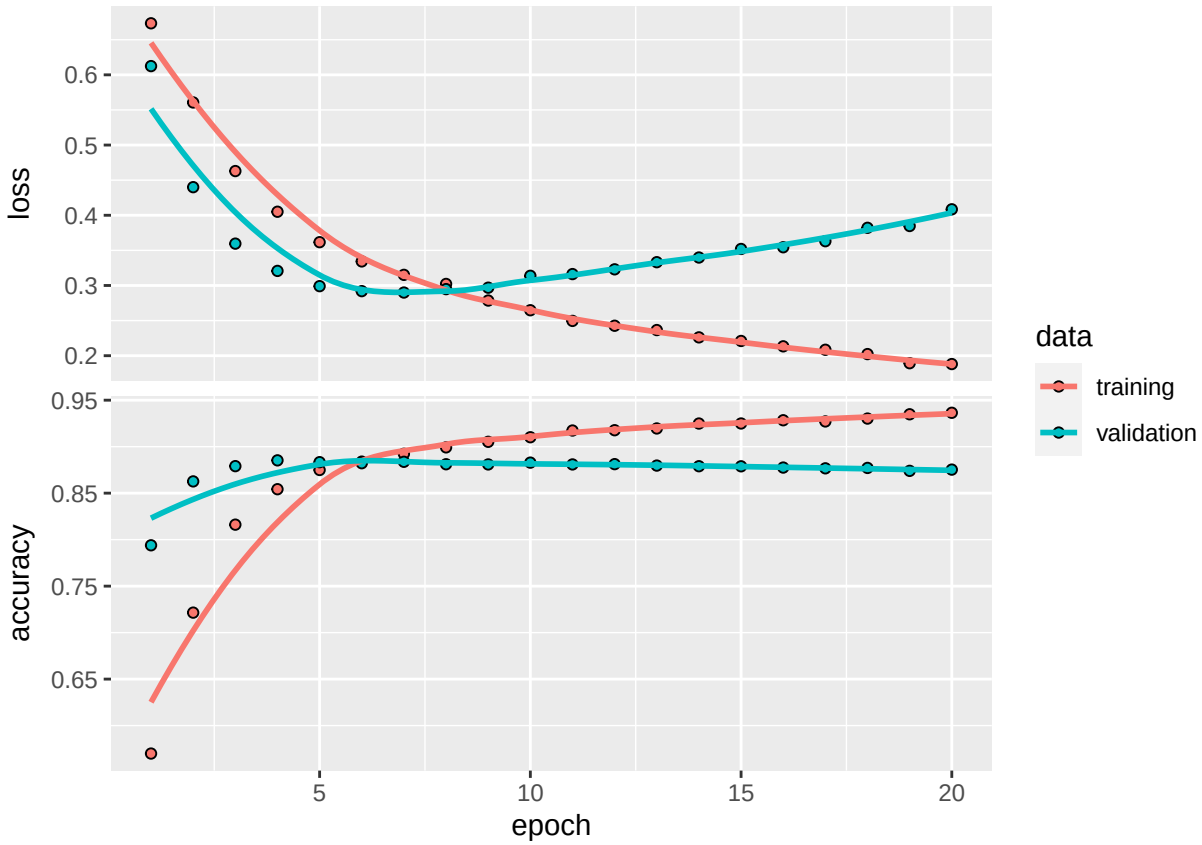
```
drop_model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = num_words) %>%
  # a new layer to specify the dropout rate
  layer_dropout(0.6) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dropout(0.6) %>%
  layer_dense(units = 1, activation = "sigmoid")

drop_model %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = list("accuracy")
)

drop_model %>% summary()
```

```
## Model: "sequential_2"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
```

```
##  dense_5 (Dense)                    (None, 16)                    80016
##
##  dropout_1 (Dropout)                (None, 16)                    0
##
##  dense_4 (Dense)                    (None, 16)                    272
##
##  dropout (Dropout)                  (None, 16)                    0
##
##  dense_3 (Dense)                    (None, 1)                     17
##
## ================================================================================
## Total params: 80,305
## Trainable params: 80,305
## Non-trainable params: 0
## --------------------------------------------------------------------------------
```

```
drop_history <- drop_model %>% fit(
  train_data,
  train_labels,
  epoch = 20,
  batch_size = 512,
  validation_data = list(test_data, test_labels),
  verbose = 2
)

plot(drop_history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```


loss
0.6
0.5
0.4
0.3
0.2
accuracy
0.95
0.85
0.75
0.65
5
10
15
20
epoch
data
training
validation

# 3 Neural Network Optimizations

## 3.1 Data & Library

```
library(readr)
library(keras)

setwd("~/Documents/Programming/R/Deep Learning with R/Datasets")
data.set <- read_csv("RegressionData.csv",
                     col_names = FALSE)
```

```
## Rows: 4898 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (11): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

## 3.2 Data Preprocessing

### 3.2.1 General Preprocessing

```
# transform dataframe to matrix
data.set <- as.matrix(data.set)
# remove column names
dimnames(data.set) <- NULL

# train test split
set.seed(123)
index <- sample(2,
                nrow(data.set),
                replace = TRUE,
                prob = c(0.8, 0.2))

x_train <- data.set[index == 1, 1:10]
x_test <- data.set[index == 2, 1:10]
y_train <- data.set[index == 1, 11]
y_test <- data.set[index == 2, 11]
```

### 3.2.2 Normalization

```
# normalizing data
mean.train <- apply(x_train, 2, mean)
sd.train <- apply(x_train, 2, sd)
x_train <- scale(x_train)
# use the normalizing parameters from training set to normalize testing set
x_test <- scale(x_test,
                center = mean.train,
                scale = sd.train)
```

### 3.3 Modeling

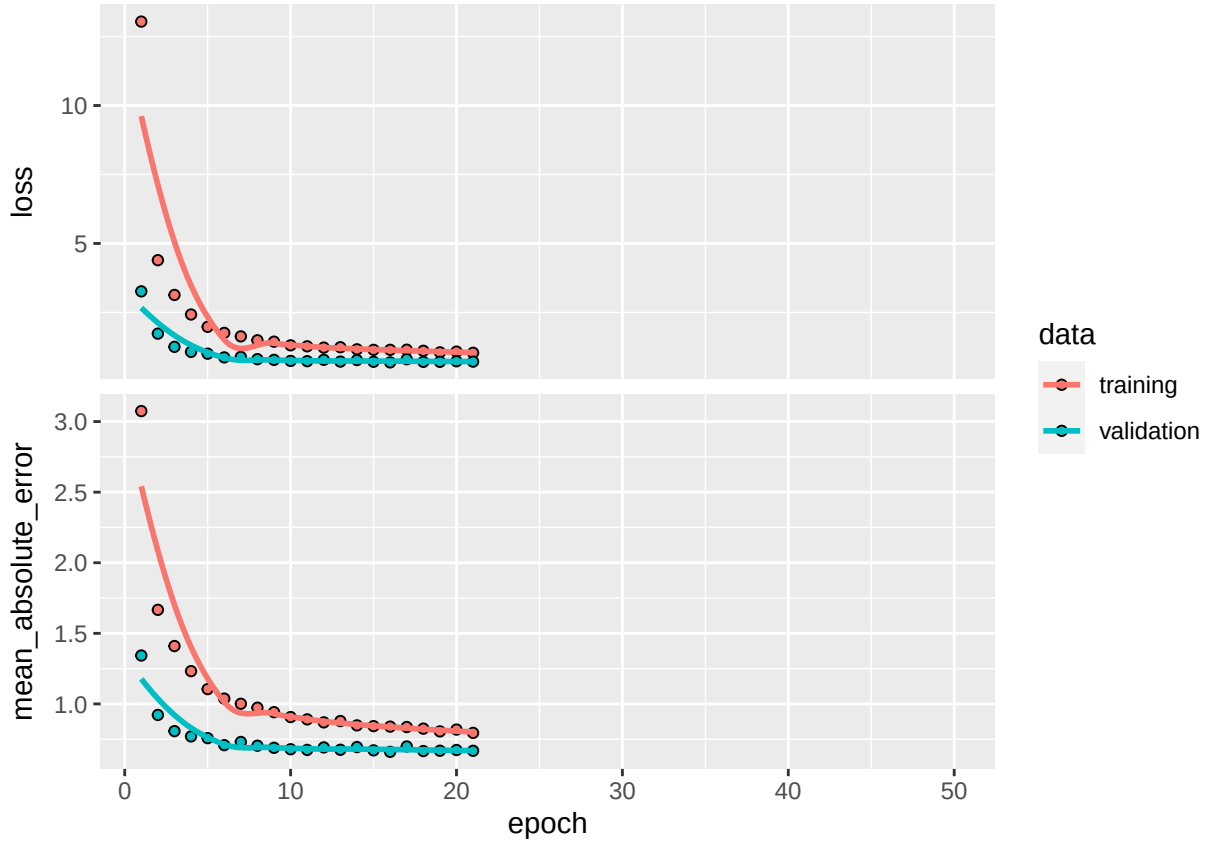
```
# creating the model
model <- keras_model_sequential() %>%
  layer_dense(units = 25,
              activation = "relu",
              input_shape = c(10)) %>%
  layer_dropout(0.2) %>%
  layer_dense(units = 25,
              activation = "relu") %>%
  layer_dropout(0.2) %>%
  layer_dense(units = 25,
              activation = "relu") %>%
  layer_dropout(0.2) %>%
  layer_dense(units = 1)

model %>% summary()
```

```
## Model: "sequential_3"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_9 (Dense)                    (None, 25)                      275
##
##  dropout_4 (Dropout)                (None, 25)                      0
##
##  dense_8 (Dense)                    (None, 25)                      650
##
##  dropout_3 (Dropout)                (None, 25)                      0
##
##  dense_7 (Dense)                    (None, 25)                      650
##
##  dropout_2 (Dropout)                (None, 25)                      0
##
##  dense_6 (Dense)                    (None, 1)                       26
##
## =========================================================================
## Total params: 1,601
## Trainable params: 1,601
## Non-trainable params: 0
## _________________________________________________________________________
```

```
# compile the model
model %>% compile(
  # the metric for propagation
  loss = "mse",
  optimizer = optimizer_rmsprop(),
  # not for propagation, but for user feedback
  # letting us know the model's performance
  metrics = c("mean_absolute_error"))

# fitting the data
model_history <- model %>%
```

```
  fit(x_train,
      y_train,
      epoch = 50,
      batch_size = 32,
      validation_split = 0.1,
      callbacks = c(callback_early_stopping(monitor = "val_mean_absolute_error",
                                            patience = 5)),
      verbose = 2)

plot(model_history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# testing the model
c(loss, mae) %<-% (model %>% evaluate(x_test, y_test, verbose = 0))
paste0("Mean Absolute Error on test set is:", mae)
```

```
## [1] "Mean Absolute Error on test set is:0.644970536231995"
```

# 4 Convolution Neural Network

## 4.1 Data & Library

```
library(keras)

# a dataset of numerous images of hand-written numbers from 0-9
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

## 4.2 Data Preprocessing

```
img_row <- dim(x_train)[2]
img_col <- dim(x_test)[3]

# images should have three channels, but the data only has two, need transformation
# because this is gray-scale image, the third channel only has one layer
x_train <- array_reshape(x_train,
                         c(nrow(x_train),
                           img_row,
                           img_col, 1))
x_test <- array_reshape(x_test,
                        c(nrow(x_test),
                          img_row,
                          img_col, 1))
input_shape <- c(img_row, img_col, 1)

# normalize the datasets by dividing the number 255
# because the color gradient is from 0(black) to 255(white)
# dividing by 255 can transform the entries to values between 0 and 1
x_train <- x_train/255
x_test <- x_test/255

# use one-hot encoding to encode the y values
# y values are labels for number 0 to 9, thus we need 10 categories
y_train <- to_categorical(y_train, num_classes = 10)
y_test <- to_categorical(y_test, num_classes = 10)
```

## 4.3 Modeling

```
model <- keras_model_sequential() %>%
  layer_conv_2d(
    # number of filters for transformation
    filters = 16,
    # size of the filters
    kernel_size = c(3,3),
    activation = 'relu',
```

```
    input_shape = input_shape) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 10,
              activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 10,
              # for categorical prediction
              activation = 'softmax')

model %>% summary()
```

```
## Model: "sequential_4"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  conv2d (Conv2D)                    (None, 26, 26, 16)              160
##
##  max_pooling2d (MaxPooling2D)       (None, 13, 13, 16)              0
##
##  dropout_6 (Dropout)                (None, 13, 13, 16)              0
##
##  flatten (Flatten)                  (None, 2704)                    0
##
##  dense_11 (Dense)                   (None, 10)                      27050
##
##  dropout_5 (Dropout)                (None, 10)                      0
##
##  dense_10 (Dense)                   (None, 10)                      110
##
## =========================================================================
## Total params: 27,320
## Trainable params: 27,320
## Non-trainable params: 0
## _________________________________________________________________________
```

```
model %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)

model %>% fit(
  x_train,
  y_train,
  batch_size = 128,
  epochs = 12,
  validation_split = 0.2
)

score <- model %>% evaluate(x_test,
                            y_test)
```

```
score
```

```
##      loss  accuracy
## 0.1689227 0.9651000
```