# Deep Learning

**Indrajith Wasala Mudiyanselage**

## Section 01

### Question 01

a) Please find the code in Section 02. When comparing models with dictionary sizes 1000 and 5000, there is a small difference in accuracy (both are quite similar to each other).

| Dictonary size | RNN model accuracy |
|---|---|
| 1000 | 0.50408 |
| 5000 | 0.50440 |

Table 1: Test accuracies for RNN model described in Section 10.9.6

b) As an improvement I have added two extra layers to the above model with ReLu activation (Please check the code in Section 02). After this modification there is an improvement in terms of accuracy.

| Dictonary size | RNN model accuracy |
|---|---|
| 1000 | 0.61397 |
| 5000 | 0.61404 |
| 10000 | 0.67408 |

Table 2: Test accuracies for improved RNN model described in Section 10.9.6

### Question 02

a) Please check the code in Section 02. After the inclusion of variable “day_of_week” there is a quite large improvement in the model $R^2$ value.

| Model | $R^2$ value |
|---|---|
| Original model | 0.408254 |
| After adding "day of week" | 0.980364 |

Table 3: Test $R^2$ value for fitted RNN model

b) I added a 32-unit dense layer with ReLu activation. The model seems to be slightly improved (Very small improvement. Check the code in Section 02).

| Model | $R^2$ value |
|---|---|
| Original model | 0.408254 |
| Improved model | 0.408319 |

Table 4: Test $R^2$ value for fitted RNN model

## Section 02 (R codes)

```
knitr::opts_chunk$set(echo = TRUE}

----eval=FALSE, include=FALSE----------------------------------------------------
## Qustion 01

# 1a)

max_features <- 5000 # or 5000
imdb <- dataset_imdb(num_words = max_features)
c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb

# a function to decode a review
word_index <- dataset_imdb_word_index()
decode_review <- function(text, word_index) {
  word <- names(word_index)
  idx <- unlist(word_index, use.names = FALSE)
  word <- c("<PAD>", "<START>", "<UNK>", "<UNUSED>", word)
  idx <- c(0:3, idx + 3)
  words <- word[match(text, idx, 2)]
  paste(words, collapse = " ")
}

library(Matrix)
one_hot <- function(sequences, dimension) {
  seqlen <- sapply(sequences, length)
  n <- length(seqlen)
  rowind <- rep(1:n, seqlen)
  colind <- unlist(sequences)
  sparseMatrix(i = rowind, j = colind,
      dims = c(n, dimension))
}

x_train_1h <- one_hot(x_train, max_features)
x_test_1h <- one_hot(x_test, max_features)

# create a validation set of size 2000 leaving 23000 for training
set.seed(3)
ival <- sample(seq(along = y_train), 2000)

### restrict the document lengths to the last L = 500 words
maxlen <- 500
x_train <- pad_sequences(x_train, maxlen = maxlen)
x_test <- pad_sequences(x_test, maxlen = maxlen)


----eval=FALSE, include=FALSE----------------------------------------------------
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(optimizer = "rmsprop",
    loss = "binary_crossentropy", metrics = c("acc"))
history <- model %>% fit(x_train, y_train, epochs = 15,
    batch_size = 128, validation_data = list(x_test, y_test))

plot(history)
predy <- predict(model, x_test) > 0.5
mean(abs(y_test == as.numeric(predy)))
```

```
----eval=FALSE, include=FALSE-----------------------------------------------------
# 1b)

max_features <- 5000 # or 5000
imdb <- dataset_imdb(num_words = max_features)
c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb

# a function to decode a review
word_index <- dataset_imdb_word_index()
decode_review <- function(text, word_index) {
  word <- names(word_index)
  idx <- unlist(word_index, use.names = FALSE)
  word <- c("<PAD>", "<START>", "<UNK>", "<UNUSED>", word)
  idx <- c(0:3, idx + 3)
  words <- word[match(text, idx, 2)]
  paste(words, collapse = " ")
}

library(Matrix)
one_hot <- function(sequences, dimension) {
  seqlen <- sapply(sequences, length)
  n <- length(seqlen)
  rowind <- rep(1:n, seqlen)
  colind <- unlist(sequences)
  sparseMatrix(i = rowind, j = colind,
      dims = c(n, dimension))
}

x_train_1h <- one_hot(x_train, max_features)
x_test_1h <- one_hot(x_test, max_features)

# create a validation set of size 2000 leaving 23000 for training
set.seed(3)
ival <- sample(seq(along = y_train), 2000)

### restrict the document lengths to the last L = 500 words
maxlen <- 500
x_train <- pad_sequences(x_train, maxlen = maxlen)
x_test <- pad_sequences(x_test, maxlen = maxlen)


----eval=FALSE, include=FALSE-----------------------------------------------------
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(optimizer = "rmsprop",
    loss = "binary_crossentropy", metrics = c("acc"))
history <- model %>% fit(x_train, y_train, epochs = 15,
    batch_size = 128, validation_data = list(x_test, y_test))

plot(history)
predy <- predict(model, x_test) > 0.5
mean(abs(y_test == as.numeric(predy)))
```

```
----eval=FALSE, include=FALSE-----------------------------------------------------
################################################################################
## Question 02

# Data Pre processing

#set up the data, and standardize each of the variables
library(ISLR2)
xdata <- data.matrix(
  NYSE[, c("DJ_return", "log_volume","log_volatility")]
)
istrain <- NYSE[, "train"]
xdata <- scale(xdata)

#functions to create lagged versions of the three time series
lagm <- function(x, k = 1) {
  n <- nrow(x)
  pad <- matrix(NA, k, ncol(x))
  rbind(pad, x[1:(n - k), ])
}

#use this function to create a data frame with all the required lags, as well as the response variable.
arframe <- data.frame(log_volume = xdata[, "log_volume"],
                      L1 = lagm(xdata, 1), L2 = lagm(xdata, 2),
                      L3 = lagm(xdata, 3), L4 = lagm(xdata, 4),
                      L5 = lagm(xdata, 5)
)

#remove NA rows, and adjust istrain accordingly
arframe <- arframe[-(1:5), ]
istrain <- istrain[-(1:5)]


----eval=FALSE, include=FALSE-----------------------------------------------------
# 2a)  fit an RNN
# reshape data to fit RNN
n <- nrow(arframe)
xrnn <- data.matrix(arframe[, -1])
xrnn <- array(xrnn, c(n, 3, 5))
xrnn <- xrnn[,, 5:1]
xrnn <- aperm(xrnn, c(1, 3, 2))

model <- keras_model_sequential() %>%
  layer_simple_rnn(units = 12,
      input_shape = list(5, 3),
      dropout = 0.1, recurrent_dropout = 0.1) %>%
  layer_dense(units = 1)
model %>% compile(optimizer = optimizer_rmsprop(),
    loss = "mse")

history <- model %>% fit(
    xrnn[istrain,, ], arframe[istrain, "log_volume"],
    batch_size = 64, epochs = 100,
    validation_data =
      list(xrnn[!istrain,, ], arframe[!istrain, "log_volume"])
  )

kpred <- predict(model, xrnn[!istrain,, ])
V0 <- var(arframe[!istrain, "log_volume"])
1 - mean((kpred - arframe[!istrain, "log_volume"])^2) / V0


----eval=FALSE, include=FALSE-----------------------------------------------------
```

```
# 2a)  fit an RNN with day_of_week added

# to include day_of_week to arframe
arframed <- data.frame(day = NYSE [-(1:5), "day_of_week"], arframe)

# reshape data to fit RNN
n <- nrow(arframed)
xrnn <- data.matrix(arframed[, -1])
xrnn <- array(xrnn, c(n, 3, 5))
xrnn <- xrnn[,, 5:1]
xrnn <- aperm(xrnn, c(1, 3, 2))

model <- keras_model_sequential() %>%
  layer_simple_rnn(units = 12,
      input_shape = list(5, 3),
      dropout = 0.1, recurrent_dropout = 0.1) %>%
  layer_dense(units = 1)
model %>% compile(optimizer = optimizer_rmsprop(),
    loss = "mse")

history <- model %>% fit(
    xrnn[istrain,, ], arframed[istrain, "log_volume"],
    batch_size = 64, epochs = 100,
    validation_data =
      list(xrnn[!istrain,, ], arframed[!istrain, "log_volume"])
  )

kpred <- predict(model, xrnn[!istrain,, ])
V0 <- var(arframed[!istrain, "log_volume"])
1 - mean((kpred - arframed[!istrain, "log_volume"])^2) / V0


----eval=FALSE, include=FALSE--------------------------------------------------
# 2b)  fit an RNN

# reshape data to fit RNN
n <- nrow(arframe)
xrnn <- data.matrix(arframe[, -1])
xrnn <- array(xrnn, c(n, 3, 5))
xrnn <- xrnn[,, 5:1]
xrnn <- aperm(xrnn, c(1, 3, 2))

model <- keras_model_sequential() %>%
  layer_simple_rnn(units = 12,
      input_shape = list(5, 3),
       dropout = 0.1, recurrent_dropout = 0.1) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1)
model %>% compile(optimizer = optimizer_rmsprop(),
    loss = "mse")

history <- model %>% fit(
    xrnn[istrain,, ], arframe[istrain, "log_volume"],
    batch_size = 64, epochs = 100,
    validation_data =
      list(xrnn[!istrain,, ], arframe[!istrain, "log_volume"])
  )

kpred <- predict(model, xrnn[!istrain,, ])
V0 <- var(arframe[!istrain, "log_volume"])
1 - mean((kpred - arframe[!istrain, "log_volume"])^2) / V0
```