STAT 6390 - Deep Learning - Mini Project 6

Nisansala Wickramasinghe

Question 1 - IMDb Document Classification

a) R code for the implementation of RNN model can be found in Section 2 part 1a). There is not much deference between accuracy of models with dictonary sizes 1000 and 5000.

Dictonary size	test accuracy of RNN model
1000	0.50167
5000	0.50232

Table 1: Test accuracies for RNN described in Section 10.9.6

b) R code for the implementation of RNN model with higher test accuracy can be found in Section 2 part 1b). I added two additional layers with sizes 64 and 32 with relu activation function.

model	test accuracy
original model with dictonary size 10000	0.50368
modified model with dictonary size 10000	0.67408

Table 2: Test accuracies for modified RNN

Question 2 - Time Series Prediction

a) R code for the implementation of RNN model can be found in Section 2 part 2a).

- Test R^2 value for original model is 0.4092936
- Test R^2 value for model after adding a variable `log_volume` is 0.9801214

b) R code for the implementation of RNN model with higher test accuracy can be found in Section 2 part 2b). I added one dense layer with 32 units and relu activation. Furthermore I changed drop out rates to 0.05. We could see slight improvement of the model.

- Test R^2 value for original model is 0.4092936
- Test R^2 value for improved model is 0.4142936

Section 2: R code

```
## ----setup, include=FALSE-------------------------------------
knitr::opts_chunk$set(echo = TRUE)
options(xtable.comment = FALSE)
knitr::opts_chunk$set(dev = 'pdf')


## ----echo=FALSE-------------------------------------------------
library(knitr)
opts_chunk$set(comment="",warning = FALSE, message=FALSE,tidy.opts=list(keep.blank.line=TRUE, width.cutoff=120),


## ----eval=FALSE, include=FALSE----------------------------------
## library(reticulate)
## Sys.setenv(RETICULATE_PYTHON= here::here('/Users/nissi_wicky/miniconda/envs/wicky/bin/python'))
## library(keras)


## ----eval=FALSE, include=FALSE----------------------------------
## ##########################################################################
## #                            Question 1
## ##########################################################################
##
## ###################### Data Pre processing ########################
##
## max_features <- 1000
## #max_features <- 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-----------------------------------
## ###################### Part 1a)  fit an LSTM RNN ########################
## 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 = 10,
##     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-----------------------------------
## ###################### Part 1b)  fit an LSTM RNN ########################
## 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 = 10,
##     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 2
## #######################################################################
##
## ###################### 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-----------------------------------
## ##################### Part 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-----------------------------------
## ##################### Part 2a)  fit an RNN with log_volume 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----------------------------------
## ###################### Part 2bbbbb)  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 = 0.05, recurrent_dropout = 0.05) %>%
##   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
```