# DSE6211 Preliminary Results

Nelson Tran

2024-02-12

Loading libraries that may be used.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(reticulate)
```

```
## Warning: package 'reticulate' was built under R version 4.3.2
```

```
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.3.2
```

```
library(tensorflow)
```

```
## Warning: package 'tensorflow' was built under R version 4.3.2
```

```
library(tidymodels)
```

```
## Warning: package 'tidymodels' was built under R version 4.3.2
```

```
## -- Attaching packages --------------------------------------- tidymodels 1.1.1 --
```

```
## v broom        1.0.5     v recipes      1.0.8
## v dials        1.2.0     v rsample      1.2.0
## v dplyr        1.1.3     v tibble       3.2.1
## v ggplot2      3.4.4     v tidyr        1.3.0
## v infer        1.0.6     v tune         1.1.2
## v modeldata    1.3.0     v workflows    1.1.3
## v parsnip      1.1.1     v workflowsets 1.0.1
## v purrr        1.0.2     v yardstick    1.3.0
```

```
## Warning: package 'dials' was built under R version 4.3.2
```

```
## Warning: package 'infer' was built under R version 4.3.2
```

```
## Warning: package 'modeldata' was built under R version 4.3.2
```

```
## Warning: package 'parsnip' was built under R version 4.3.2
```

```
## Warning: package 'recipes' was built under R version 4.3.2
```

```
## Warning: package 'rsample' was built under R version 4.3.2
```

```
## Warning: package 'tune' was built under R version 4.3.2
```

```
## Warning: package 'workflows' was built under R version 4.3.2
## Warning: package 'workflowsets' was built under R version 4.3.2
## Warning: package 'yardstick' was built under R version 4.3.2
## -- Conflicts --------------------------------------- tidymodels_conflicts() --
## x purrr::discard()         masks scales::discard()
## x dplyr::filter()          masks stats::filter()
## x yardstick::get_weights() masks keras::get_weights()
## x dplyr::lag()             masks stats::lag()
## x recipes::step()          masks stats::step()
## * Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.2
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity
## The following object is masked from 'package:purrr':
##
##     lift
## The following object is masked from 'package:tensorflow':
##
##     train
```

Data cleaning

```
data <- read.csv("./project_data.csv", header = TRUE)

#removing Booking ID column from our data.
data <- subset(data, select = -c(0,1))

#seeing the length of each room type reserved
table(data$room_type_reserved)
```

```
##
## room_type1 room_type2 room_type3 room_type4 room_type5 room_type6 room_type7
##      28105        692          7       6049        263        964        158
```

```
#From the table, we can see that 'room_type1' is largely abundant compared to the other rooms.
#turning this variable into a factor  'room_type1" and 'other
room_type <- data %>%
  group_by(room_type_reserved) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  select(room_type_reserved) %>%
  top_n(-1)
```

```
## Selecting by room_type_reserved
```

```
data$room_type_reserved <- ifelse(data$room_type_reserved %in% room_type$room_type_reserved,
                                  data$room_type_reserved,
                                  "other")
unique(data$room_type_reserved)
```

```
## [1] "room_type1" "other"
```

```
unique(data$type_of_meal_plan)
```

```
## [1] "meal_plan_1"  "not_selected" "meal_plan_2"  "meal_plan_3"
```

```
unique(data$market_segment_type)
```

```
## [1] "offline"       "online"        "corporate"     "aviation"
## [5] "complementary"
```

```
#I will leave 'type_of_meal_plan' and 'market_segment_type' as is but will still need to convert to a f
```

```
#Converting dates from "arrival_date" to seasons
data2 <- data
data2 <- data.frame(
  arrival_date = seq(as.Date("2017-01-01"), as.Date("2018-12-31"), by = "days")
)
get_season <- function(month) {
  case_when(
    month %in% 3:5 ~ "Spring",
    month %in% 6:8 ~ "Summer",
    month %in% 9:11 ~ "Fall",
    TRUE ~ "Winter"
  )
}
data2 <- data2 %>%
  mutate(month = month(arrival_date),
         season = get_season(month))
data <- data %>%
  mutate(arrival_date = get_season(month(arrival_date)))

unique(data$arrival_date)
```

```
## [1] "Fall"   "Winter" "Spring" "Summer"
```

Character variables will be changed to factors

```
data$type_of_meal_plan <- factor(data$type_of_meal_plan)
data$room_type_reserved <- factor(data$room_type_reserved)
data$arrival_date <- factor(data$arrival_date)
data$market_segment_type <- factor(data$market_segment_type)
#cancelled is = 1, not_cancelled = 0
data$booking_status <- as.integer(data$booking_status == "canceled")
```

Onehot_encoder

```
training_ind <- createDataPartition(data$booking_status,
                                    p = 0.75,
                                    list = FALSE,
                                    times = 1)
training_set <- data[training_ind,]
test_set <- data[-training_ind,]
```

```
onehot_encoder <- dummyVars(~ type_of_meal_plan + room_type_reserved +
                              arrival_date + market_segment_type,
                            training_set[, c("type_of_meal_plan", "room_type_reserved",
                                             "arrival_date", "market_segment_type")],
                            levelsOnly = TRUE,
                            fullRank = TRUE)

onehot_enc_training <- predict(onehot_encoder,
                               training_set[, c("type_of_meal_plan", "room_type_reserved",
                                             "arrival_date", "market_segment_type")])

training_set <- cbind(training_set, onehot_enc_training)

onehot_encoder <- dummyVars(~ type_of_meal_plan + room_type_reserved +
                              arrival_date + market_segment_type,
                            test_set[, c("type_of_meal_plan", "room_type_reserved",
                                             "arrival_date", "market_segment_type")],
                            levelsOnly = TRUE,
                            fullRank = TRUE)

onehot_enc_test <- predict(onehot_encoder,
                               test_set[, c("type_of_meal_plan", "room_type_reserved",
                                             "arrival_date", "market_segment_type")])

test_set <- cbind(test_set, onehot_enc_test)

cols_to_exclude <- c(5,7,9,10,16)

test_set[, -(cols_to_exclude)] <- scale(test_set[, -c(cols_to_exclude)],
                            center = apply(training_set[, -c(cols_to_exclude)], 2, mean),
                            scale = apply(training_set[, -c(cols_to_exclude)], 2, sd))
training_set[, -c(cols_to_exclude)] <- scale(training_set[, -c(cols_to_exclude)])

training_features <- array(data = unlist(training_set[, -c(cols_to_exclude)]),
                                         dim = c(nrow(training_set), 27))
training_labels <- array(data = unlist(training_set[, 16]),
                         dim = c(nrow(training_set)))

test_features <- array(data = unlist(test_set[, -c(cols_to_exclude)]),
                       dim = c(nrow(test_set), 27))
test_labels <- array(data = unlist(test_set[, 16]),
                     dim = c(nrow(test_set)))
```

Loading our environment

```
use_virtualenv("my_tf_workspace")
```

Testing different architectures to see which one will fit the best.

model_1 (10 units)

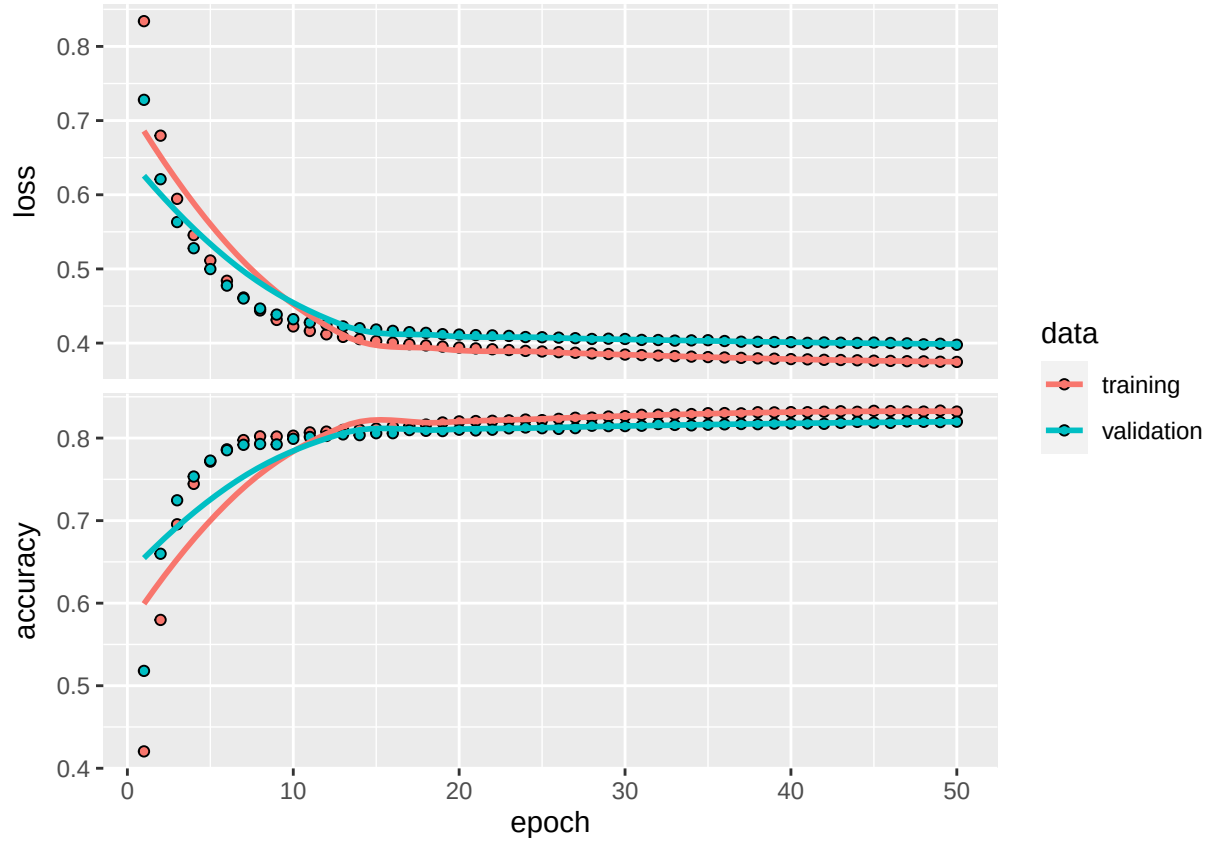
```
model_1 <- keras_model_sequential(list(
  layer_dense(units = 10, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
```

```
compile(model_1,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")

history_1 <- fit(model_1, training_features, training_labels,
                 epochs = 50 , batch_size = 512, validation_split = 0.33)
```

```
## Epoch 1/50
## 36/36 - 1s - loss: 0.8341 - accuracy: 0.4205 - val_loss: 0.7280 - val_accuracy: 0.5179 - 507ms/epoch
## Epoch 2/50
## 36/36 - 0s - loss: 0.6797 - accuracy: 0.5797 - val_loss: 0.6210 - val_accuracy: 0.6599 - 69ms/epoch
## Epoch 3/50
## 36/36 - 0s - loss: 0.5945 - accuracy: 0.6956 - val_loss: 0.5632 - val_accuracy: 0.7246 - 49ms/epoch
## Epoch 4/50
## 36/36 - 0s - loss: 0.5457 - accuracy: 0.7445 - val_loss: 0.5279 - val_accuracy: 0.7534 - 45ms/epoch
## Epoch 5/50
## 36/36 - 0s - loss: 0.5114 - accuracy: 0.7715 - val_loss: 0.4998 - val_accuracy: 0.7727 - 47ms/epoch
## Epoch 6/50
## 36/36 - 0s - loss: 0.4840 - accuracy: 0.7864 - val_loss: 0.4775 - val_accuracy: 0.7853 - 46ms/epoch
## Epoch 7/50
## 36/36 - 0s - loss: 0.4615 - accuracy: 0.7976 - val_loss: 0.4601 - val_accuracy: 0.7919 - 46ms/epoch
## Epoch 8/50
## 36/36 - 0s - loss: 0.4440 - accuracy: 0.8021 - val_loss: 0.4465 - val_accuracy: 0.7929 - 43ms/epoch
## Epoch 9/50
## 36/36 - 0s - loss: 0.4313 - accuracy: 0.8019 - val_loss: 0.4384 - val_accuracy: 0.7923 - 47ms/epoch
## Epoch 10/50
## 36/36 - 0s - loss: 0.4225 - accuracy: 0.8029 - val_loss: 0.4322 - val_accuracy: 0.7991 - 43ms/epoch
## Epoch 11/50
## 36/36 - 0s - loss: 0.4164 - accuracy: 0.8069 - val_loss: 0.4277 - val_accuracy: 0.8013 - 47ms/epoch
## Epoch 12/50
## 36/36 - 0s - loss: 0.4118 - accuracy: 0.8078 - val_loss: 0.4250 - val_accuracy: 0.8026 - 44ms/epoch
## Epoch 13/50
## 36/36 - 0s - loss: 0.4084 - accuracy: 0.8099 - val_loss: 0.4225 - val_accuracy: 0.8045 - 48ms/epoch
## Epoch 14/50
## 36/36 - 0s - loss: 0.4054 - accuracy: 0.8102 - val_loss: 0.4201 - val_accuracy: 0.8036 - 46ms/epoch
## Epoch 15/50
## 36/36 - 0s - loss: 0.4026 - accuracy: 0.8115 - val_loss: 0.4183 - val_accuracy: 0.8059 - 46ms/epoch
## Epoch 16/50
## 36/36 - 0s - loss: 0.4005 - accuracy: 0.8140 - val_loss: 0.4165 - val_accuracy: 0.8059 - 44ms/epoch
## Epoch 17/50
## 36/36 - 0s - loss: 0.3985 - accuracy: 0.8148 - val_loss: 0.4147 - val_accuracy: 0.8097 - 47ms/epoch
## Epoch 18/50
## 36/36 - 0s - loss: 0.3968 - accuracy: 0.8164 - val_loss: 0.4138 - val_accuracy: 0.8088 - 58ms/epoch
## Epoch 19/50
## 36/36 - 0s - loss: 0.3953 - accuracy: 0.8189 - val_loss: 0.4120 - val_accuracy: 0.8085 - 44ms/epoch
## Epoch 20/50
## 36/36 - 0s - loss: 0.3940 - accuracy: 0.8202 - val_loss: 0.4115 - val_accuracy: 0.8101 - 43ms/epoch
## Epoch 21/50
## 36/36 - 0s - loss: 0.3928 - accuracy: 0.8206 - val_loss: 0.4108 - val_accuracy: 0.8093 - 44ms/epoch
## Epoch 22/50
## 36/36 - 0s - loss: 0.3917 - accuracy: 0.8210 - val_loss: 0.4104 - val_accuracy: 0.8101 - 43ms/epoch
## Epoch 23/50
## 36/36 - 0s - loss: 0.3906 - accuracy: 0.8215 - val_loss: 0.4097 - val_accuracy: 0.8118 - 43ms/epoch
```

```
## Epoch 24/50
## 36/36 - 0s - loss: 0.3895 - accuracy: 0.8225 - val_loss: 0.4082 - val_accuracy: 0.8128 - 44ms/epoch -
## Epoch 25/50
## 36/36 - 0s - loss: 0.3887 - accuracy: 0.8217 - val_loss: 0.4079 - val_accuracy: 0.8119 - 44ms/epoch -
## Epoch 26/50
## 36/36 - 0s - loss: 0.3878 - accuracy: 0.8233 - val_loss: 0.4073 - val_accuracy: 0.8111 - 42ms/epoch -
## Epoch 27/50
## 36/36 - 0s - loss: 0.3870 - accuracy: 0.8246 - val_loss: 0.4066 - val_accuracy: 0.8120 - 43ms/epoch -
## Epoch 28/50
## 36/36 - 0s - loss: 0.3861 - accuracy: 0.8248 - val_loss: 0.4056 - val_accuracy: 0.8149 - 44ms/epoch -
## Epoch 29/50
## 36/36 - 0s - loss: 0.3853 - accuracy: 0.8261 - val_loss: 0.4059 - val_accuracy: 0.8145 - 45ms/epoch -
## Epoch 30/50
## 36/36 - 0s - loss: 0.3845 - accuracy: 0.8266 - val_loss: 0.4056 - val_accuracy: 0.8145 - 43ms/epoch -
## Epoch 31/50
## 36/36 - 0s - loss: 0.3838 - accuracy: 0.8282 - val_loss: 0.4043 - val_accuracy: 0.8149 - 44ms/epoch -
## Epoch 32/50
## 36/36 - 0s - loss: 0.3831 - accuracy: 0.8284 - val_loss: 0.4041 - val_accuracy: 0.8171 - 44ms/epoch -
## Epoch 33/50
## 36/36 - 0s - loss: 0.3825 - accuracy: 0.8285 - val_loss: 0.4033 - val_accuracy: 0.8159 - 44ms/epoch -
## Epoch 34/50
## 36/36 - 0s - loss: 0.3818 - accuracy: 0.8291 - val_loss: 0.4036 - val_accuracy: 0.8154 - 43ms/epoch -
## Epoch 35/50
## 36/36 - 0s - loss: 0.3811 - accuracy: 0.8301 - val_loss: 0.4039 - val_accuracy: 0.8162 - 43ms/epoch -
## Epoch 36/50
## 36/36 - 0s - loss: 0.3806 - accuracy: 0.8302 - val_loss: 0.4028 - val_accuracy: 0.8166 - 44ms/epoch -
## Epoch 37/50
## 36/36 - 0s - loss: 0.3801 - accuracy: 0.8300 - val_loss: 0.4019 - val_accuracy: 0.8171 - 42ms/epoch -
## Epoch 38/50
## 36/36 - 0s - loss: 0.3794 - accuracy: 0.8313 - val_loss: 0.4017 - val_accuracy: 0.8168 - 45ms/epoch -
## Epoch 39/50
## 36/36 - 0s - loss: 0.3790 - accuracy: 0.8311 - val_loss: 0.4014 - val_accuracy: 0.8175 - 44ms/epoch -
## Epoch 40/50
## 36/36 - 0s - loss: 0.3784 - accuracy: 0.8315 - val_loss: 0.4013 - val_accuracy: 0.8175 - 44ms/epoch -
## Epoch 41/50
## 36/36 - 0s - loss: 0.3781 - accuracy: 0.8313 - val_loss: 0.4003 - val_accuracy: 0.8177 - 43ms/epoch -
## Epoch 42/50
## 36/36 - 0s - loss: 0.3774 - accuracy: 0.8320 - val_loss: 0.4009 - val_accuracy: 0.8173 - 43ms/epoch -
## Epoch 43/50
## 36/36 - 0s - loss: 0.3772 - accuracy: 0.8325 - val_loss: 0.4005 - val_accuracy: 0.8184 - 44ms/epoch -
## Epoch 44/50
## 36/36 - 0s - loss: 0.3768 - accuracy: 0.8318 - val_loss: 0.4002 - val_accuracy: 0.8196 - 45ms/epoch -
## Epoch 45/50
## 36/36 - 0s - loss: 0.3763 - accuracy: 0.8330 - val_loss: 0.4007 - val_accuracy: 0.8187 - 44ms/epoch -
## Epoch 46/50
## 36/36 - 0s - loss: 0.3760 - accuracy: 0.8327 - val_loss: 0.4002 - val_accuracy: 0.8184 - 43ms/epoch -
## Epoch 47/50
## 36/36 - 0s - loss: 0.3756 - accuracy: 0.8323 - val_loss: 0.3997 - val_accuracy: 0.8201 - 44ms/epoch -
## Epoch 48/50
## 36/36 - 0s - loss: 0.3754 - accuracy: 0.8321 - val_loss: 0.3983 - val_accuracy: 0.8197 - 42ms/epoch -
## Epoch 49/50
## 36/36 - 0s - loss: 0.3749 - accuracy: 0.8333 - val_loss: 0.3991 - val_accuracy: 0.8194 - 44ms/epoch -
## Epoch 50/50
## 36/36 - 0s - loss: 0.3746 - accuracy: 0.8321 - val_loss: 0.3978 - val_accuracy: 0.8198 - 43ms/epoch -
```

```
plot(history_1)
```




model_2 (20 units, 10 units respectively)

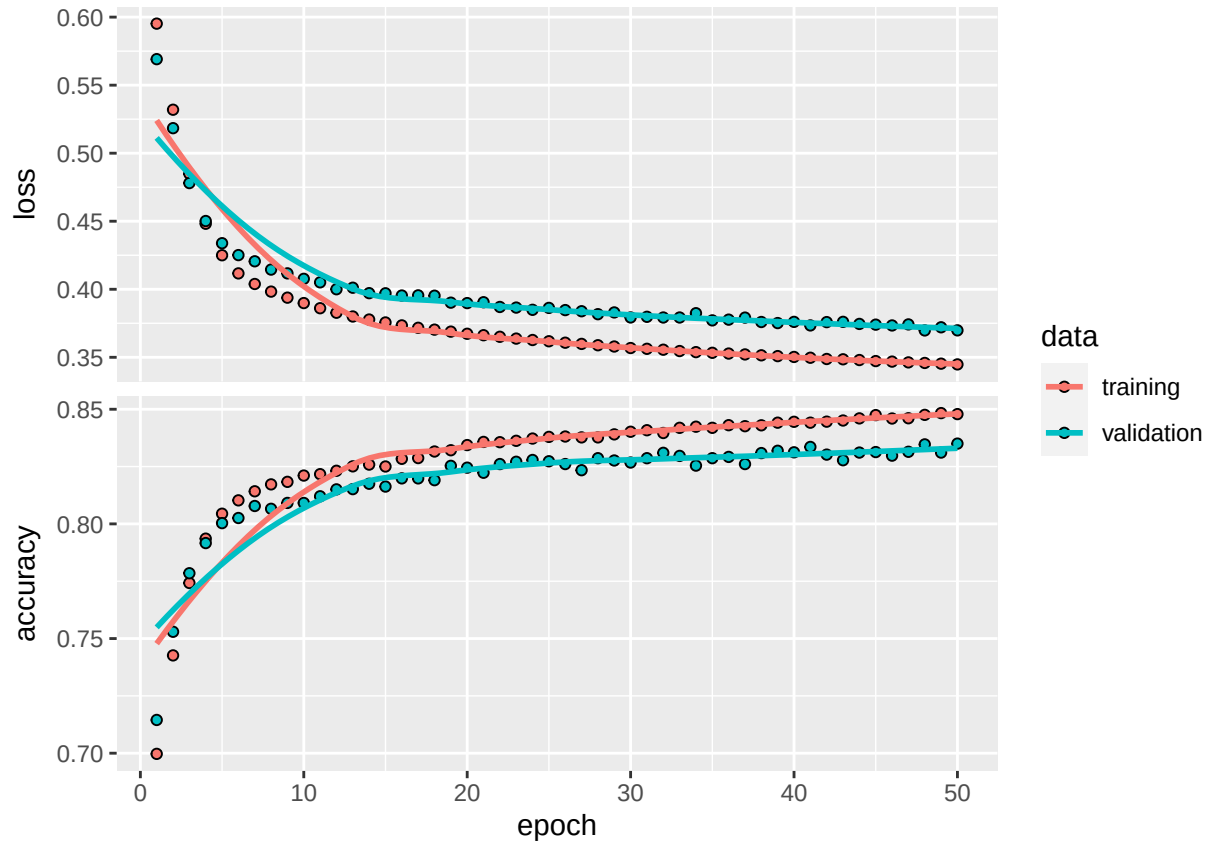
```
model_2 <- keras_model_sequential(list(
  layer_dense(units = 20, activation = "relu"),
  layer_dense(units = 10, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model_2,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")

history_2 <- fit(model_2, training_features, training_labels,
                 epochs = 50, batch_size = 512, validation_split = 0.33)
```

```
## Epoch 1/50
## 36/36 - 0s - loss: 0.5952 - accuracy: 0.6997 - val_loss: 0.5691 - val_accuracy: 0.7145 - 389ms/epoch
## Epoch 2/50
## 36/36 - 0s - loss: 0.5319 - accuracy: 0.7427 - val_loss: 0.5184 - val_accuracy: 0.7530 - 45ms/epoch
## Epoch 3/50
## 36/36 - 0s - loss: 0.4849 - accuracy: 0.7742 - val_loss: 0.4780 - val_accuracy: 0.7785 - 45ms/epoch
## Epoch 4/50
## 36/36 - 0s - loss: 0.4482 - accuracy: 0.7936 - val_loss: 0.4501 - val_accuracy: 0.7916 - 45ms/epoch
## Epoch 5/50
## 36/36 - 0s - loss: 0.4250 - accuracy: 0.8044 - val_loss: 0.4338 - val_accuracy: 0.8003 - 44ms/epoch
```

```
## Epoch 6/50
## 36/36 - 0s - loss: 0.4117 - accuracy: 0.8103 - val_loss: 0.4251 - val_accuracy: 0.8026 - 45ms/epoch -
## Epoch 7/50
## 36/36 - 0s - loss: 0.4039 - accuracy: 0.8142 - val_loss: 0.4206 - val_accuracy: 0.8078 - 48ms/epoch -
## Epoch 8/50
## 36/36 - 0s - loss: 0.3983 - accuracy: 0.8172 - val_loss: 0.4145 - val_accuracy: 0.8066 - 45ms/epoch -
## Epoch 9/50
## 36/36 - 0s - loss: 0.3938 - accuracy: 0.8183 - val_loss: 0.4117 - val_accuracy: 0.8091 - 46ms/epoch -
## Epoch 10/50
## 36/36 - 0s - loss: 0.3898 - accuracy: 0.8211 - val_loss: 0.4077 - val_accuracy: 0.8091 - 45ms/epoch -
## Epoch 11/50
## 36/36 - 0s - loss: 0.3861 - accuracy: 0.8217 - val_loss: 0.4051 - val_accuracy: 0.8120 - 46ms/epoch -
## Epoch 12/50
## 36/36 - 0s - loss: 0.3827 - accuracy: 0.8232 - val_loss: 0.4000 - val_accuracy: 0.8151 - 45ms/epoch -
## Epoch 13/50
## 36/36 - 0s - loss: 0.3800 - accuracy: 0.8251 - val_loss: 0.4011 - val_accuracy: 0.8152 - 45ms/epoch -
## Epoch 14/50
## 36/36 - 0s - loss: 0.3778 - accuracy: 0.8259 - val_loss: 0.3971 - val_accuracy: 0.8175 - 44ms/epoch -
## Epoch 15/50
## 36/36 - 0s - loss: 0.3756 - accuracy: 0.8250 - val_loss: 0.3970 - val_accuracy: 0.8163 - 45ms/epoch -
## Epoch 16/50
## 36/36 - 0s - loss: 0.3734 - accuracy: 0.8284 - val_loss: 0.3954 - val_accuracy: 0.8198 - 45ms/epoch -
## Epoch 17/50
## 36/36 - 0s - loss: 0.3717 - accuracy: 0.8287 - val_loss: 0.3955 - val_accuracy: 0.8198 - 45ms/epoch -
## Epoch 18/50
## 36/36 - 0s - loss: 0.3703 - accuracy: 0.8316 - val_loss: 0.3953 - val_accuracy: 0.8191 - 44ms/epoch -
## Epoch 19/50
## 36/36 - 0s - loss: 0.3689 - accuracy: 0.8321 - val_loss: 0.3902 - val_accuracy: 0.8253 - 46ms/epoch -
## Epoch 20/50
## 36/36 - 0s - loss: 0.3673 - accuracy: 0.8344 - val_loss: 0.3898 - val_accuracy: 0.8245 - 44ms/epoch -
## Epoch 21/50
## 36/36 - 0s - loss: 0.3662 - accuracy: 0.8357 - val_loss: 0.3905 - val_accuracy: 0.8223 - 45ms/epoch -
## Epoch 22/50
## 36/36 - 0s - loss: 0.3651 - accuracy: 0.8356 - val_loss: 0.3870 - val_accuracy: 0.8261 - 46ms/epoch -
## Epoch 23/50
## 36/36 - 0s - loss: 0.3637 - accuracy: 0.8362 - val_loss: 0.3865 - val_accuracy: 0.8271 - 57ms/epoch -
## Epoch 24/50
## 36/36 - 0s - loss: 0.3627 - accuracy: 0.8372 - val_loss: 0.3849 - val_accuracy: 0.8279 - 45ms/epoch -
## Epoch 25/50
## 36/36 - 0s - loss: 0.3618 - accuracy: 0.8379 - val_loss: 0.3863 - val_accuracy: 0.8273 - 45ms/epoch -
## Epoch 26/50
## 36/36 - 0s - loss: 0.3607 - accuracy: 0.8381 - val_loss: 0.3847 - val_accuracy: 0.8261 - 45ms/epoch -
## Epoch 27/50
## 36/36 - 0s - loss: 0.3598 - accuracy: 0.8377 - val_loss: 0.3838 - val_accuracy: 0.8234 - 45ms/epoch -
## Epoch 28/50
## 36/36 - 0s - loss: 0.3589 - accuracy: 0.8377 - val_loss: 0.3817 - val_accuracy: 0.8287 - 45ms/epoch -
## Epoch 29/50
## 36/36 - 0s - loss: 0.3579 - accuracy: 0.8390 - val_loss: 0.3830 - val_accuracy: 0.8276 - 44ms/epoch -
## Epoch 30/50
## 36/36 - 0s - loss: 0.3568 - accuracy: 0.8402 - val_loss: 0.3793 - val_accuracy: 0.8268 - 45ms/epoch -
## Epoch 31/50
## 36/36 - 0s - loss: 0.3562 - accuracy: 0.8408 - val_loss: 0.3798 - val_accuracy: 0.8287 - 47ms/epoch -
## Epoch 32/50
## 36/36 - 0s - loss: 0.3556 - accuracy: 0.8396 - val_loss: 0.3792 - val_accuracy: 0.8310 - 45ms/epoch -
```

```
## Epoch 33/50
## 36/36 - 0s - loss: 0.3546 - accuracy: 0.8419 - val_loss: 0.3793 - val_accuracy: 0.8297 - 44ms/epoch -
## Epoch 34/50
## 36/36 - 0s - loss: 0.3538 - accuracy: 0.8424 - val_loss: 0.3824 - val_accuracy: 0.8254 - 49ms/epoch -
## Epoch 35/50
## 36/36 - 0s - loss: 0.3533 - accuracy: 0.8419 - val_loss: 0.3771 - val_accuracy: 0.8287 - 47ms/epoch -
## Epoch 36/50
## 36/36 - 0s - loss: 0.3527 - accuracy: 0.8430 - val_loss: 0.3777 - val_accuracy: 0.8292 - 46ms/epoch -
## Epoch 37/50
## 36/36 - 0s - loss: 0.3521 - accuracy: 0.8426 - val_loss: 0.3792 - val_accuracy: 0.8261 - 46ms/epoch -
## Epoch 38/50
## 36/36 - 0s - loss: 0.3515 - accuracy: 0.8430 - val_loss: 0.3759 - val_accuracy: 0.8309 - 46ms/epoch -
## Epoch 39/50
## 36/36 - 0s - loss: 0.3509 - accuracy: 0.8441 - val_loss: 0.3752 - val_accuracy: 0.8319 - 44ms/epoch -
## Epoch 40/50
## 36/36 - 0s - loss: 0.3503 - accuracy: 0.8445 - val_loss: 0.3760 - val_accuracy: 0.8312 - 45ms/epoch -
## Epoch 41/50
## 36/36 - 0s - loss: 0.3497 - accuracy: 0.8441 - val_loss: 0.3734 - val_accuracy: 0.8336 - 44ms/epoch -
## Epoch 42/50
## 36/36 - 0s - loss: 0.3488 - accuracy: 0.8446 - val_loss: 0.3759 - val_accuracy: 0.8302 - 46ms/epoch -
## Epoch 43/50
## 36/36 - 0s - loss: 0.3485 - accuracy: 0.8451 - val_loss: 0.3761 - val_accuracy: 0.8278 - 45ms/epoch -
## Epoch 44/50
## 36/36 - 0s - loss: 0.3480 - accuracy: 0.8460 - val_loss: 0.3745 - val_accuracy: 0.8311 - 45ms/epoch -
## Epoch 45/50
## 36/36 - 0s - loss: 0.3472 - accuracy: 0.8474 - val_loss: 0.3740 - val_accuracy: 0.8313 - 45ms/epoch -
## Epoch 46/50
## 36/36 - 0s - loss: 0.3468 - accuracy: 0.8460 - val_loss: 0.3734 - val_accuracy: 0.8298 - 44ms/epoch -
## Epoch 47/50
## 36/36 - 0s - loss: 0.3463 - accuracy: 0.8461 - val_loss: 0.3742 - val_accuracy: 0.8314 - 45ms/epoch -
## Epoch 48/50
## 36/36 - 0s - loss: 0.3458 - accuracy: 0.8475 - val_loss: 0.3697 - val_accuracy: 0.8347 - 45ms/epoch -
## Epoch 49/50
## 36/36 - 0s - loss: 0.3453 - accuracy: 0.8483 - val_loss: 0.3720 - val_accuracy: 0.8311 - 44ms/epoch -
## Epoch 50/50
## 36/36 - 0s - loss: 0.3448 - accuracy: 0.8479 - val_loss: 0.3698 - val_accuracy: 0.8350 - 46ms/epoch -
```

```
plot(history_2)
```




model_3 (50 units, and 25 units respectively)

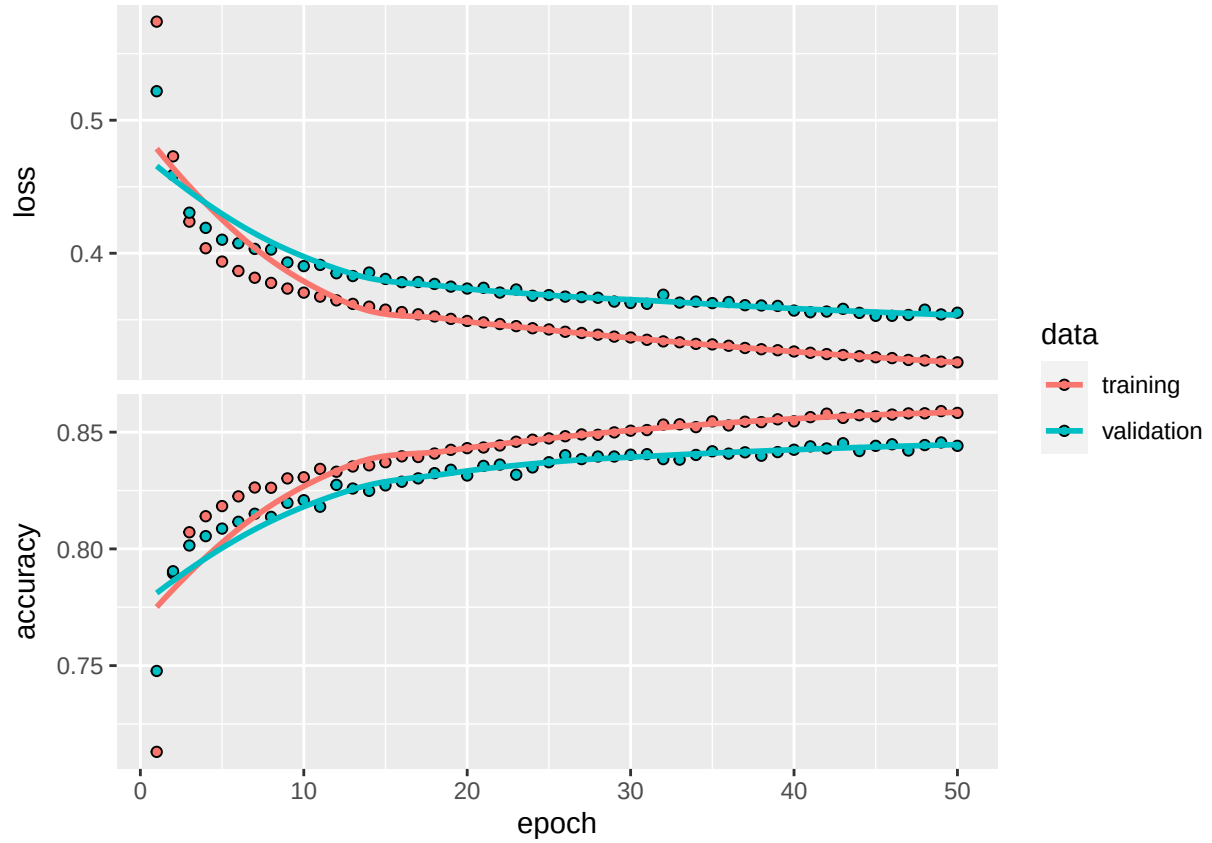
```
model_3 <- keras_model_sequential(list(
  layer_dense(units = 50, activation = "relu"),
  layer_dense(units = 25, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model_3,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")

history_3 <- fit(model_3, training_features, training_labels,
                 epochs = 50, batch_size = 512, validation_split = 0.33)
```

```
## Epoch 1/50
## 36/36 - 0s - loss: 0.5741 - accuracy: 0.7131 - val_loss: 0.5218 - val_accuracy: 0.7477 - 391ms/epoch
## Epoch 2/50
## 36/36 - 0s - loss: 0.4728 - accuracy: 0.7896 - val_loss: 0.4589 - val_accuracy: 0.7904 - 48ms/epoch
## Epoch 3/50
## 36/36 - 0s - loss: 0.4238 - accuracy: 0.8071 - val_loss: 0.4305 - val_accuracy: 0.8014 - 47ms/epoch
## Epoch 4/50
## 36/36 - 0s - loss: 0.4038 - accuracy: 0.8140 - val_loss: 0.4191 - val_accuracy: 0.8055 - 46ms/epoch
## Epoch 5/50
## 36/36 - 0s - loss: 0.3937 - accuracy: 0.8183 - val_loss: 0.4102 - val_accuracy: 0.8087 - 47ms/epoch
## Epoch 6/50
## 36/36 - 0s - loss: 0.3867 - accuracy: 0.8225 - val_loss: 0.4075 - val_accuracy: 0.8116 - 48ms/epoch
```

```
## Epoch 7/50
## 36/36 - 0s - loss: 0.3816 - accuracy: 0.8263 - val_loss: 0.4033 - val_accuracy: 0.8151 - 48ms/epoch -
## Epoch 8/50
## 36/36 - 0s - loss: 0.3777 - accuracy: 0.8262 - val_loss: 0.4028 - val_accuracy: 0.8137 - 47ms/epoch -
## Epoch 9/50
## 36/36 - 0s - loss: 0.3735 - accuracy: 0.8302 - val_loss: 0.3931 - val_accuracy: 0.8197 - 48ms/epoch -
## Epoch 10/50
## 36/36 - 0s - loss: 0.3704 - accuracy: 0.8307 - val_loss: 0.3903 - val_accuracy: 0.8208 - 48ms/epoch -
## Epoch 11/50
## 36/36 - 0s - loss: 0.3674 - accuracy: 0.8343 - val_loss: 0.3912 - val_accuracy: 0.8181 - 48ms/epoch -
## Epoch 12/50
## 36/36 - 0s - loss: 0.3646 - accuracy: 0.8330 - val_loss: 0.3850 - val_accuracy: 0.8274 - 47ms/epoch -
## Epoch 13/50
## 36/36 - 0s - loss: 0.3620 - accuracy: 0.8353 - val_loss: 0.3829 - val_accuracy: 0.8259 - 48ms/epoch -
## Epoch 14/50
## 36/36 - 0s - loss: 0.3598 - accuracy: 0.8358 - val_loss: 0.3855 - val_accuracy: 0.8249 - 48ms/epoch -
## Epoch 15/50
## 36/36 - 0s - loss: 0.3576 - accuracy: 0.8371 - val_loss: 0.3807 - val_accuracy: 0.8272 - 47ms/epoch -
## Epoch 16/50
## 36/36 - 0s - loss: 0.3558 - accuracy: 0.8396 - val_loss: 0.3783 - val_accuracy: 0.8288 - 47ms/epoch -
## Epoch 17/50
## 36/36 - 0s - loss: 0.3541 - accuracy: 0.8394 - val_loss: 0.3784 - val_accuracy: 0.8302 - 48ms/epoch -
## Epoch 18/50
## 36/36 - 0s - loss: 0.3525 - accuracy: 0.8408 - val_loss: 0.3769 - val_accuracy: 0.8324 - 47ms/epoch -
## Epoch 19/50
## 36/36 - 0s - loss: 0.3507 - accuracy: 0.8424 - val_loss: 0.3749 - val_accuracy: 0.8339 - 48ms/epoch -
## Epoch 20/50
## 36/36 - 0s - loss: 0.3491 - accuracy: 0.8432 - val_loss: 0.3735 - val_accuracy: 0.8314 - 47ms/epoch -
## Epoch 21/50
## 36/36 - 0s - loss: 0.3480 - accuracy: 0.8434 - val_loss: 0.3740 - val_accuracy: 0.8356 - 48ms/epoch -
## Epoch 22/50
## 36/36 - 0s - loss: 0.3468 - accuracy: 0.8443 - val_loss: 0.3704 - val_accuracy: 0.8361 - 47ms/epoch -
## Epoch 23/50
## 36/36 - 0s - loss: 0.3452 - accuracy: 0.8459 - val_loss: 0.3728 - val_accuracy: 0.8318 - 47ms/epoch -
## Epoch 24/50
## 36/36 - 0s - loss: 0.3436 - accuracy: 0.8468 - val_loss: 0.3680 - val_accuracy: 0.8349 - 48ms/epoch -
## Epoch 25/50
## 36/36 - 0s - loss: 0.3427 - accuracy: 0.8473 - val_loss: 0.3686 - val_accuracy: 0.8371 - 48ms/epoch -
## Epoch 26/50
## 36/36 - 0s - loss: 0.3410 - accuracy: 0.8483 - val_loss: 0.3674 - val_accuracy: 0.8401 - 48ms/epoch -
## Epoch 27/50
## 36/36 - 0s - loss: 0.3401 - accuracy: 0.8490 - val_loss: 0.3670 - val_accuracy: 0.8385 - 47ms/epoch -
## Epoch 28/50
## 36/36 - 0s - loss: 0.3388 - accuracy: 0.8489 - val_loss: 0.3665 - val_accuracy: 0.8396 - 48ms/epoch -
## Epoch 29/50
## 36/36 - 0s - loss: 0.3373 - accuracy: 0.8499 - val_loss: 0.3637 - val_accuracy: 0.8396 - 48ms/epoch -
## Epoch 30/50
## 36/36 - 0s - loss: 0.3367 - accuracy: 0.8506 - val_loss: 0.3626 - val_accuracy: 0.8404 - 48ms/epoch -
## Epoch 31/50
## 36/36 - 0s - loss: 0.3349 - accuracy: 0.8509 - val_loss: 0.3620 - val_accuracy: 0.8406 - 48ms/epoch -
## Epoch 32/50
## 36/36 - 0s - loss: 0.3337 - accuracy: 0.8533 - val_loss: 0.3688 - val_accuracy: 0.8385 - 49ms/epoch -
## Epoch 33/50
## 36/36 - 0s - loss: 0.3331 - accuracy: 0.8534 - val_loss: 0.3629 - val_accuracy: 0.8382 - 48ms/epoch -
```

```
## Epoch 34/50
## 36/36 - 0s - loss: 0.3319 - accuracy: 0.8522 - val_loss: 0.3637 - val_accuracy: 0.8402 - 48ms/epoch -
## Epoch 35/50
## 36/36 - 0s - loss: 0.3315 - accuracy: 0.8546 - val_loss: 0.3625 - val_accuracy: 0.8418 - 60ms/epoch -
## Epoch 36/50
## 36/36 - 0s - loss: 0.3305 - accuracy: 0.8529 - val_loss: 0.3634 - val_accuracy: 0.8408 - 47ms/epoch -
## Epoch 37/50
## 36/36 - 0s - loss: 0.3288 - accuracy: 0.8545 - val_loss: 0.3610 - val_accuracy: 0.8414 - 49ms/epoch -
## Epoch 38/50
## 36/36 - 0s - loss: 0.3279 - accuracy: 0.8543 - val_loss: 0.3608 - val_accuracy: 0.8399 - 49ms/epoch -
## Epoch 39/50
## 36/36 - 0s - loss: 0.3271 - accuracy: 0.8555 - val_loss: 0.3604 - val_accuracy: 0.8415 - 49ms/epoch -
## Epoch 40/50
## 36/36 - 0s - loss: 0.3262 - accuracy: 0.8546 - val_loss: 0.3569 - val_accuracy: 0.8425 - 65ms/epoch -
## Epoch 41/50
## 36/36 - 0s - loss: 0.3252 - accuracy: 0.8564 - val_loss: 0.3557 - val_accuracy: 0.8439 - 51ms/epoch -
## Epoch 42/50
## 36/36 - 0s - loss: 0.3242 - accuracy: 0.8579 - val_loss: 0.3562 - val_accuracy: 0.8430 - 49ms/epoch -
## Epoch 43/50
## 36/36 - 0s - loss: 0.3235 - accuracy: 0.8561 - val_loss: 0.3582 - val_accuracy: 0.8453 - 48ms/epoch -
## Epoch 44/50
## 36/36 - 0s - loss: 0.3226 - accuracy: 0.8573 - val_loss: 0.3552 - val_accuracy: 0.8419 - 49ms/epoch -
## Epoch 45/50
## 36/36 - 0s - loss: 0.3218 - accuracy: 0.8568 - val_loss: 0.3530 - val_accuracy: 0.8441 - 48ms/epoch -
## Epoch 46/50
## 36/36 - 0s - loss: 0.3212 - accuracy: 0.8575 - val_loss: 0.3530 - val_accuracy: 0.8448 - 49ms/epoch -
## Epoch 47/50
## 36/36 - 0s - loss: 0.3198 - accuracy: 0.8580 - val_loss: 0.3536 - val_accuracy: 0.8421 - 47ms/epoch -
## Epoch 48/50
## 36/36 - 0s - loss: 0.3194 - accuracy: 0.8581 - val_loss: 0.3576 - val_accuracy: 0.8445 - 48ms/epoch -
## Epoch 49/50
## 36/36 - 0s - loss: 0.3185 - accuracy: 0.8590 - val_loss: 0.3540 - val_accuracy: 0.8456 - 47ms/epoch -
## Epoch 50/50
## 36/36 - 0s - loss: 0.3180 - accuracy: 0.8583 - val_loss: 0.3553 - val_accuracy: 0.8441 - 49ms/epoch -
```

```
plot(history_3)
```




Model_4 (75 units, 50 units, 25 units respectively)

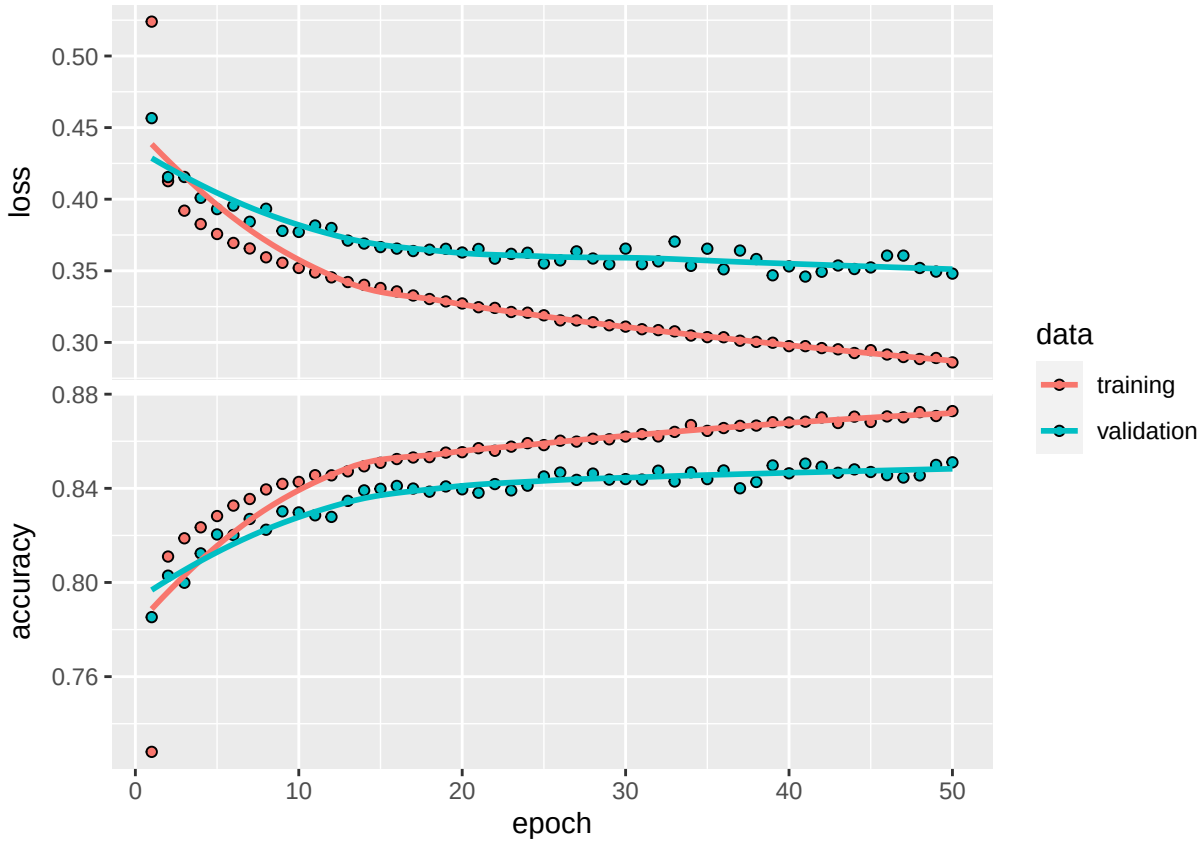
```
model_4 <- keras_model_sequential(list(
  layer_dense(units = 75, activation = "relu"),
  layer_dense(units = 50, activation = "relu"),
  layer_dense(units = 25, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))
compile(model_4,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")

history_4 <- fit(model_4, training_features, training_labels,
                 epochs = 50, batch_size = 512, validation_split = 0.33)
```

```
## Epoch 1/50
## 36/36 - 0s - loss: 0.5240 - accuracy: 0.7280 - val_loss: 0.4566 - val_accuracy: 0.7853 - 447ms/epoch
## Epoch 2/50
## 36/36 - 0s - loss: 0.4126 - accuracy: 0.8110 - val_loss: 0.4155 - val_accuracy: 0.8029 - 56ms/epoch
## Epoch 3/50
## 36/36 - 0s - loss: 0.3920 - accuracy: 0.8188 - val_loss: 0.4155 - val_accuracy: 0.7999 - 56ms/epoch
## Epoch 4/50
## 36/36 - 0s - loss: 0.3827 - accuracy: 0.8234 - val_loss: 0.4010 - val_accuracy: 0.8124 - 55ms/epoch
## Epoch 5/50
## 36/36 - 0s - loss: 0.3757 - accuracy: 0.8282 - val_loss: 0.3931 - val_accuracy: 0.8204 - 56ms/epoch
## Epoch 6/50
```

```
## 36/36 - 0s - loss: 0.3695 - accuracy: 0.8327 - val_loss: 0.3955 - val_accuracy: 0.8202 - 56ms/epoch -
## Epoch 7/50
## 36/36 - 0s - loss: 0.3656 - accuracy: 0.8355 - val_loss: 0.3842 - val_accuracy: 0.8270 - 55ms/epoch -
## Epoch 8/50
## 36/36 - 0s - loss: 0.3594 - accuracy: 0.8395 - val_loss: 0.3933 - val_accuracy: 0.8224 - 54ms/epoch -
## Epoch 9/50
## 36/36 - 0s - loss: 0.3556 - accuracy: 0.8419 - val_loss: 0.3779 - val_accuracy: 0.8302 - 55ms/epoch -
## Epoch 10/50
## 36/36 - 0s - loss: 0.3520 - accuracy: 0.8427 - val_loss: 0.3771 - val_accuracy: 0.8298 - 55ms/epoch -
## Epoch 11/50
## 36/36 - 0s - loss: 0.3488 - accuracy: 0.8456 - val_loss: 0.3816 - val_accuracy: 0.8285 - 53ms/epoch -
## Epoch 12/50
## 36/36 - 0s - loss: 0.3454 - accuracy: 0.8455 - val_loss: 0.3799 - val_accuracy: 0.8279 - 53ms/epoch -
## Epoch 13/50
## 36/36 - 0s - loss: 0.3422 - accuracy: 0.8472 - val_loss: 0.3711 - val_accuracy: 0.8347 - 55ms/epoch -
## Epoch 14/50
## 36/36 - 0s - loss: 0.3402 - accuracy: 0.8494 - val_loss: 0.3690 - val_accuracy: 0.8391 - 54ms/epoch -
## Epoch 15/50
## 36/36 - 0s - loss: 0.3380 - accuracy: 0.8508 - val_loss: 0.3666 - val_accuracy: 0.8398 - 53ms/epoch -
## Epoch 16/50
## 36/36 - 0s - loss: 0.3356 - accuracy: 0.8524 - val_loss: 0.3655 - val_accuracy: 0.8410 - 53ms/epoch -
## Epoch 17/50
## 36/36 - 0s - loss: 0.3328 - accuracy: 0.8531 - val_loss: 0.3638 - val_accuracy: 0.8399 - 58ms/epoch -
## Epoch 18/50
## 36/36 - 0s - loss: 0.3303 - accuracy: 0.8533 - val_loss: 0.3647 - val_accuracy: 0.8386 - 57ms/epoch -
## Epoch 19/50
## 36/36 - 0s - loss: 0.3286 - accuracy: 0.8552 - val_loss: 0.3654 - val_accuracy: 0.8408 - 53ms/epoch -
## Epoch 20/50
## 36/36 - 0s - loss: 0.3272 - accuracy: 0.8553 - val_loss: 0.3628 - val_accuracy: 0.8396 - 55ms/epoch -
## Epoch 21/50
## 36/36 - 0s - loss: 0.3246 - accuracy: 0.8570 - val_loss: 0.3653 - val_accuracy: 0.8381 - 55ms/epoch -
## Epoch 22/50
## 36/36 - 0s - loss: 0.3241 - accuracy: 0.8560 - val_loss: 0.3585 - val_accuracy: 0.8418 - 53ms/epoch -
## Epoch 23/50
## 36/36 - 0s - loss: 0.3213 - accuracy: 0.8577 - val_loss: 0.3618 - val_accuracy: 0.8391 - 55ms/epoch -
## Epoch 24/50
## 36/36 - 0s - loss: 0.3207 - accuracy: 0.8592 - val_loss: 0.3626 - val_accuracy: 0.8411 - 54ms/epoch -
## Epoch 25/50
## 36/36 - 0s - loss: 0.3190 - accuracy: 0.8584 - val_loss: 0.3551 - val_accuracy: 0.8450 - 54ms/epoch -
## Epoch 26/50
## 36/36 - 0s - loss: 0.3154 - accuracy: 0.8602 - val_loss: 0.3573 - val_accuracy: 0.8467 - 53ms/epoch -
## Epoch 27/50
## 36/36 - 0s - loss: 0.3153 - accuracy: 0.8598 - val_loss: 0.3636 - val_accuracy: 0.8436 - 55ms/epoch -
## Epoch 28/50
## 36/36 - 0s - loss: 0.3141 - accuracy: 0.8611 - val_loss: 0.3586 - val_accuracy: 0.8463 - 54ms/epoch -
## Epoch 29/50
## 36/36 - 0s - loss: 0.3120 - accuracy: 0.8608 - val_loss: 0.3546 - val_accuracy: 0.8437 - 53ms/epoch -
## Epoch 30/50
## 36/36 - 0s - loss: 0.3110 - accuracy: 0.8620 - val_loss: 0.3655 - val_accuracy: 0.8439 - 54ms/epoch -
## Epoch 31/50
## 36/36 - 0s - loss: 0.3092 - accuracy: 0.8630 - val_loss: 0.3547 - val_accuracy: 0.8437 - 55ms/epoch -
## Epoch 32/50
## 36/36 - 0s - loss: 0.3086 - accuracy: 0.8620 - val_loss: 0.3566 - val_accuracy: 0.8475 - 54ms/epoch -
## Epoch 33/50
```

```
## 36/36 - 0s - loss: 0.3077 - accuracy: 0.8640 - val_loss: 0.3704 - val_accuracy: 0.8429 - 55ms/epoch -
## Epoch 34/50
## 36/36 - 0s - loss: 0.3048 - accuracy: 0.8669 - val_loss: 0.3535 - val_accuracy: 0.8468 - 55ms/epoch -
## Epoch 35/50
## 36/36 - 0s - loss: 0.3037 - accuracy: 0.8644 - val_loss: 0.3654 - val_accuracy: 0.8439 - 53ms/epoch -
## Epoch 36/50
## 36/36 - 0s - loss: 0.3036 - accuracy: 0.8656 - val_loss: 0.3510 - val_accuracy: 0.8476 - 55ms/epoch -
## Epoch 37/50
## 36/36 - 0s - loss: 0.3012 - accuracy: 0.8665 - val_loss: 0.3641 - val_accuracy: 0.8400 - 54ms/epoch -
## Epoch 38/50
## 36/36 - 0s - loss: 0.3004 - accuracy: 0.8667 - val_loss: 0.3582 - val_accuracy: 0.8427 - 58ms/epoch -
## Epoch 39/50
## 36/36 - 0s - loss: 0.2998 - accuracy: 0.8681 - val_loss: 0.3469 - val_accuracy: 0.8497 - 55ms/epoch -
## Epoch 40/50
## 36/36 - 0s - loss: 0.2974 - accuracy: 0.8680 - val_loss: 0.3531 - val_accuracy: 0.8464 - 56ms/epoch -
## Epoch 41/50
## 36/36 - 0s - loss: 0.2974 - accuracy: 0.8683 - val_loss: 0.3460 - val_accuracy: 0.8505 - 55ms/epoch -
## Epoch 42/50
## 36/36 - 0s - loss: 0.2960 - accuracy: 0.8701 - val_loss: 0.3493 - val_accuracy: 0.8492 - 53ms/epoch -
## Epoch 43/50
## 36/36 - 0s - loss: 0.2952 - accuracy: 0.8678 - val_loss: 0.3537 - val_accuracy: 0.8466 - 55ms/epoch -
## Epoch 44/50
## 36/36 - 0s - loss: 0.2926 - accuracy: 0.8704 - val_loss: 0.3513 - val_accuracy: 0.8480 - 65ms/epoch -
## Epoch 45/50
## 36/36 - 0s - loss: 0.2946 - accuracy: 0.8682 - val_loss: 0.3523 - val_accuracy: 0.8469 - 54ms/epoch -
## Epoch 46/50
## 36/36 - 0s - loss: 0.2915 - accuracy: 0.8706 - val_loss: 0.3606 - val_accuracy: 0.8456 - 53ms/epoch -
## Epoch 47/50
## 36/36 - 0s - loss: 0.2898 - accuracy: 0.8702 - val_loss: 0.3606 - val_accuracy: 0.8446 - 53ms/epoch -
## Epoch 48/50
## 36/36 - 0s - loss: 0.2884 - accuracy: 0.8724 - val_loss: 0.3520 - val_accuracy: 0.8455 - 55ms/epoch -
## Epoch 49/50
## 36/36 - 0s - loss: 0.2891 - accuracy: 0.8707 - val_loss: 0.3495 - val_accuracy: 0.8499 - 54ms/epoch -
## Epoch 50/50
## 36/36 - 0s - loss: 0.2860 - accuracy: 0.8728 - val_loss: 0.3481 - val_accuracy: 0.8511 - 55ms/epoch -
```

```
plot(history_4)
```




Aspects on the neural networks for why/how they were chosen

number of layers: I wanted to see how adding and removing layers would affect how the neural network learns and it appears that 2 hidden layers gives the best results out of the models used above.

number of units: Increasing the number of nodes in each layer also shows positive signs of improvement, although it seems like 50 and 25 nodes respectively is the sweet spot. Although this can be changed if needed during further testing.

activation functions: For the hidden layers, we are using the "relu" function between it helps the network learn non-linear relationships. For the output layer, since we are dealing with a binary classification problem (0,1) we are using the sigmoid function.

loss function: This is a binary problem so we will use "binary_crossentropy"

optimizer: RMSprop seems like a good optimizer to use but after doing a tad bit of research, ADAM could be a possible choice of an optimizer as well.

Comparing the learning curves with each other, I feel like there seems to be both underfitting models (1,2) and over fitting models (3,4). Based on this I think sticking with 2 hidden layers will be the best fit while keeping the units for each layer around 30.

As there was already a scale added to the data, I think that possibly selecting the most important features could be beneficial to the networks. I can try and look for outliers that may have a negative impact to the overall model performance. Cross-validation and hyperparameter tuning which can include learning rates.

What we can do to improve for final report

Excellent work on the Preliminary Results, Nelson! Since all of the requirements for this assignment have been satisfied, the most important requirements to focus on for the Final Report are: -Try at least one additional dense feed-forward neural network with a different architecture. Try to obtain a model that is

capable of overfitting (e.g., increase the number of hidden layers and/or units per hidden layer). Chapter 5 of the Deep Learning with R textbook covers several aspects of building deep learning models in practice. Section 5.3 is particular focused on improving the model fit using methods such as varying the number of units and layers. The reason we want to obtain a model that is capable of overfitting is because this model is flexible enough for the problem at hand. Then, we implement methods to prevent overfitting, such as early-stopping (i.e., stop at the epoch number where overfitting starts to occur), dropout, or regularization. -Evaluate and compare the dense feed-forward neural networks using ROC curves, AUC, and calibration curves. -Discuss the findings from the above analyses, as well as how the model will be used in practice. -Discuss future research/steps forward. The steps forward can include data that may be worthwhile including in future analyses. Please let me know if you have any questions! Best, Peter