Final Report_Simonsen

Steven Simonsen

2024-10-15

Final Report

Below is the Final Report for the binary classification problem where the label is to determine hotel bookings for ABC Hotels that are at a high risk of cancellation. The Final Report was a continuation and improvement of the models created in the Preliminary Results. Additionally, the Preliminary Results are included for reference underneath the Final Report.

Similar to the Preliminary Results, I began the process by importing the data and labeling the positive class (i.e. cancelled bookings) to "1", and the non-cancelled bookings to "0". As identified in my Analytic Plan, and implemented in my Preliminary results, I completed data engineering on the arrival_date predictor to extract the year and season. Then, I again split the data into a 75/25% split for training and test sets, respectively.

Again, I then created a recipe as before, and converted the data to arrays to prepare for the implementation of neural networks. I created three additional models to further improve my overfitting results from the Preliminary Results, and then improve generalization through the implementation of various techniques discussed below.

Model 6 Creation: Model 6 was a continuation from the best performing model in terms of overfitting from the Preliminary Results. To recall, model 5 previously performed the best overfitting results. I therefore used the same number of hidden layers with the same number of units in each layer, but changed the optimizer from rmsprop to an adam_optimizer to see if this would improve my results. I was fairly optimistic that this could potentially yield even better overfitting results. My thought process was that the adam_optimizer uses bias correction by using both the moving average of the gradients (first moment) and the squared gradients (second moment), while rmsprop only uses the moving average of the squared gradients to scale the learning rate. However, the outcome of only changing the optimizer yielded results that were not significantly different than using rmsprop. Therefore, due to the simplicity of rmsprop in comparison, I decided to keep this as my optimizer.

Model 7 Creation: Going back to my rmsprop optimizer, I instead decided to focus on the number of units I had in my hidden layers within the model. I increased the number of hidden layers from 87 in the first hidden layer, to 348, or 12 times the number of predictors in the baked training set. I also increased the subsequent number of units in each hidden layer, and this resulted in a model that overfit very well. As seen in the calibration curves, the training loss ended at about a value of 0.03, while my training accuracy ended at about a value of 0.98. The validation loss curve continuously climbed higher, while the validation accuracy leveled off early at about 0.85. This was encouraging, and was by far my best result. Of course, my roc curve looked average at best, and my auc score, while very good, could have been better. I decided to use this model as my basis to begin improving my generalization, while also cutting back the overfitting to increase the results on the test set for the roc/auc.

Model 8 Creation: Here, I kept the results of model 7, and implemented both early stopping and dropout to help to prevent overfitting. In my model, I implemented dropout between each of the hidden layers since I had a large number of units in each layer. I also adopted a dropout rate of 0.5 to encourage a good balance between regularizing the model and allowing it to learn useful patterns. Then, before fitting the model, I implemented early stopping and passed it into the fitting of the model as a callback value. I provided the early stopping with a patience parameter of 50, and while this may be a bit high, I wanted to give the model a chance to improve over the course of several epochs since I had initially implemented 750. The model stopped early after 245 epochs, and yielded me my results.

Model 8 evaluation: After fitting the model to my training data, I then made predictions and bound the predictions to the test set. I then used this data to graph a ROC curve, and generate the AUC score. The ROC curve looked very good, with a strong rise in TPR (True Positive Rate), before leveling off near 1 at the upper left side of the graph. Additionally the AUC (Area Under Curve) score was fantastic, yielding a result of about 0.93. I was happy with the result of this model, especially considering I was able to improve my generalization results from the techniques mentioned.

How Can this Be Used and Next Steps: Were this model to be implemented into Production in the real world, ABC Hotels could used it as a means to predict whether a customer booking was at a high risk for cancellation by feeding the model new data on a regular basis. ABC Hotels may target those customer flagged as high cancellation risk to implement advanced marketing tactics, promotional offers, and more to encourage the customers to keep their bookings. Though this model yielded great results, the next steps as a Data Scientist would be to continuously monitor and update the model. For example, perhaps new data points could be collected to improve the scores even more. Or, on another note, perhaps some of the features may prove to be less indicative of cancellation risk over time and be omitted from the results as noise. Practically speaking, ABC Hotels could greatly benefit from this model, and therefore increase their profits in doing so.

```r
set.seed(42)
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.2.0 --

## v broom        1.0.6     v recipes      1.1.0
## v dials        1.3.0     v rsample      1.2.1
## v dplyr        1.1.4     v tibble       3.2.1
## v ggplot2      3.5.1     v tidyr        1.3.1
## v infer        1.0.7     v tune         1.2.1
## v modeldata    1.4.0     v workflows    1.1.4
## v parsnip      1.2.1     v workflowsets 1.1.0
## v purrr        1.0.2     v yardstick    1.3.1

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
## * Learn how to get started at https://www.tidymodels.org/start/
```

```r
library(reticulate)
library(tensorflow)
library(keras3)
```

```
##
## Attaching package: 'keras3'

## The following objects are masked from 'package:tensorflow':
##
##     set_random_seed, shape

## The following object is masked from 'package:yardstick':
##
##     get_weights
```

```r
library(MESS)

getwd()
```

```
## [1] "C:/Users/steve/OneDrive/Documents/School/DSE6211/Project"
```

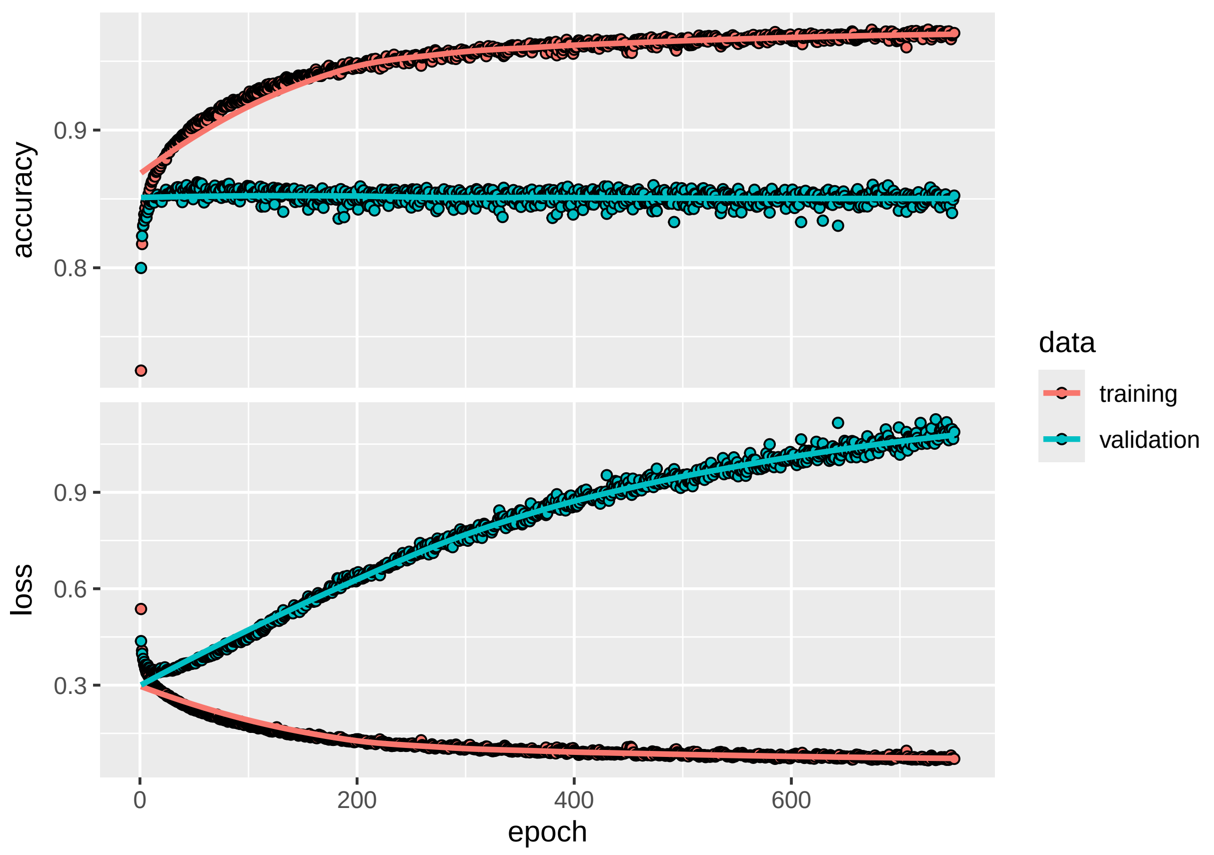
```r
setwd("C:\\Users\\steve\\OneDrive\\Documents\\School\\DSE6211\\Project")

data <- read.csv("project_data.csv")

data <- data[, -1]
data$booking_status <- ifelse(data$booking_status == "canceled", 1, 0)

#Function creation to extract seasons from arrival date
data$arrival_date <- as.Date(data$arrival_date)
data$month <- format(data$arrival_date, "%m")
data$year <- format(data$arrival_date, "%Y")

get_season <- function(month) {
  month <- as.numeric(month)
  if (month %in% c(12, 1, 2)) {
    return("Winter")
  } else if (month %in% c(3, 4, 5)) {
    return("Spring")
  } else if (month %in% c(6, 7, 8)) {
    return("Summer")
  } else if (month %in% c(9, 10, 11)) {
    return("Autumn")
  }
}

data$season <- sapply(data$month, get_season)

data <- data[, -9]
data <- data[, -16]

set.seed(42)
data_split <- initial_split(data, strata = "booking_status", prop = 0.75)

training_set <- training(data_split)
test_set  <- testing(data_split)

booking_recipe <-
  recipe(
    booking_status ~ .,
    data = training_set
  ) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  prep(training = training_set, retain = TRUE)

training_set_baked <- bake(booking_recipe, new_data = training_set)
test_set_baked <- bake(booking_recipe, new_data = test_set)



training_features <- array(data = unlist(training_set_baked[, -12]),
                           dim = c(nrow(training_set_baked), 28))
training_labels <- array(data = unlist(training_set_baked[, 12]),
                         dim = c(nrow(training_set_baked)))

test_features <- array(data = unlist(test_set_baked[, -12]),
                       dim = c(nrow(test_set_baked), 28))
test_labels <- array(data = unlist(test_set_baked[, 12]),
                     dim = c(nrow(test_set_baked)))

use_virtualenv("my_tf_workspace")
```

Model 6 - Adam optimizer instead of rmsprop with last model since this achieved the most overfitting

```
set.seed(42)
model6 <- keras_model_sequential() %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dense(units = 70, activation = "relu") %>%
  layer_dense(units = 53, activation = "relu") %>%
  layer_dense(units = 36, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")


compile(model6,
        optimizer = optimizer_adam(),
        loss = "binary_crossentropy",
        metrics = "accuracy")



random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```
set.seed(42)
history6 <- fit(model6, training_features, training_labels,
               epochs = 750, batch_size = 512, validation_split = 0.33)
```

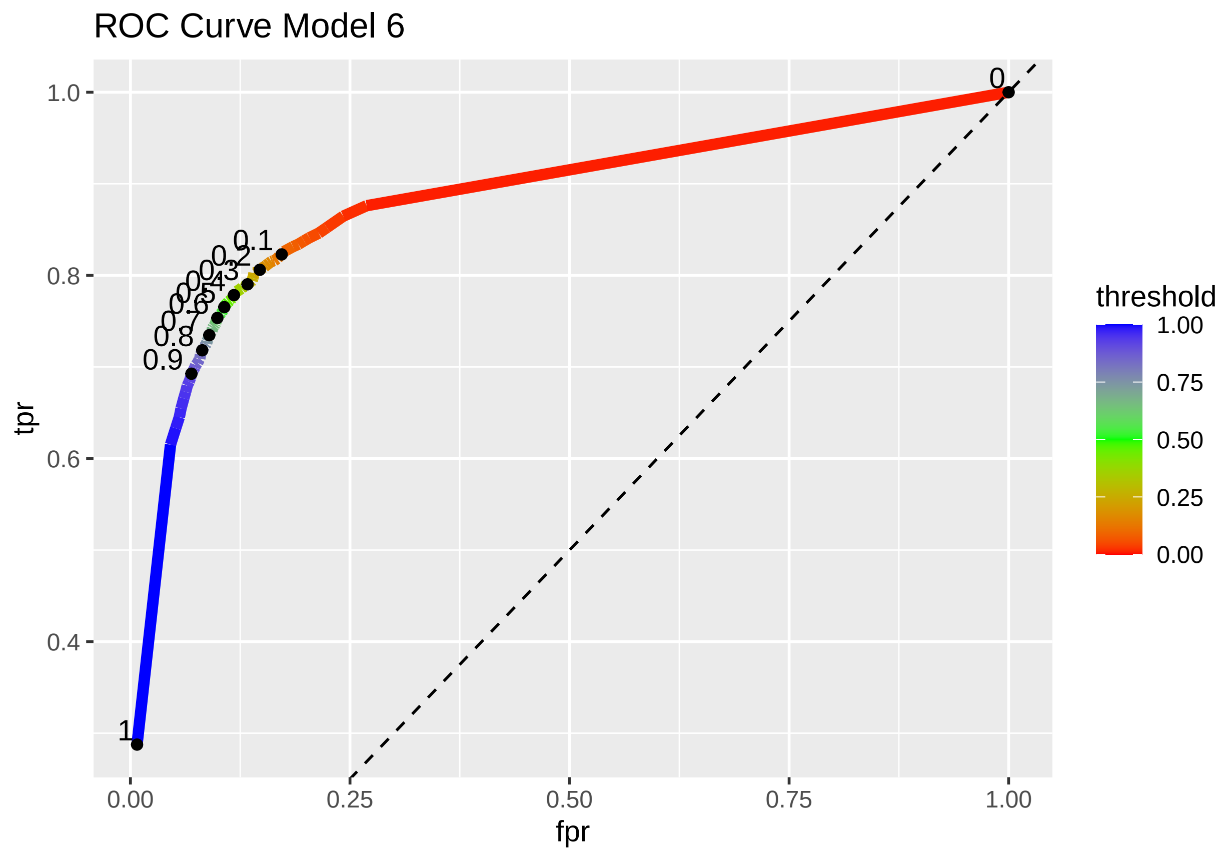
```
set.seed(42)
plot(history6)
```



```r
predictions6 <- predict(model6, test_features)
```

```
## 284/284 - 0s - 794us/step
```

```r
test_results6 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions6)
  )

roc_data6 <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data6$threshold) {

  over_threshold <- test_results6[test_results6$p_1 >= i, ]

  fpr <- sum(over_threshold$booking_status==0)/sum(test_results6$booking_status==0)
  roc_data6[roc_data6$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results6$booking_status==1)
  roc_data6[roc_data6$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data6, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data6[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data6[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 6")
```



```r
auc6 <- auc(x = roc_data6$fpr, y = roc_data6$tpr, type = "spline")
```

```
## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique
## 'x' values
```

```r
auc6
```

```
## [1] 0.8934974
```

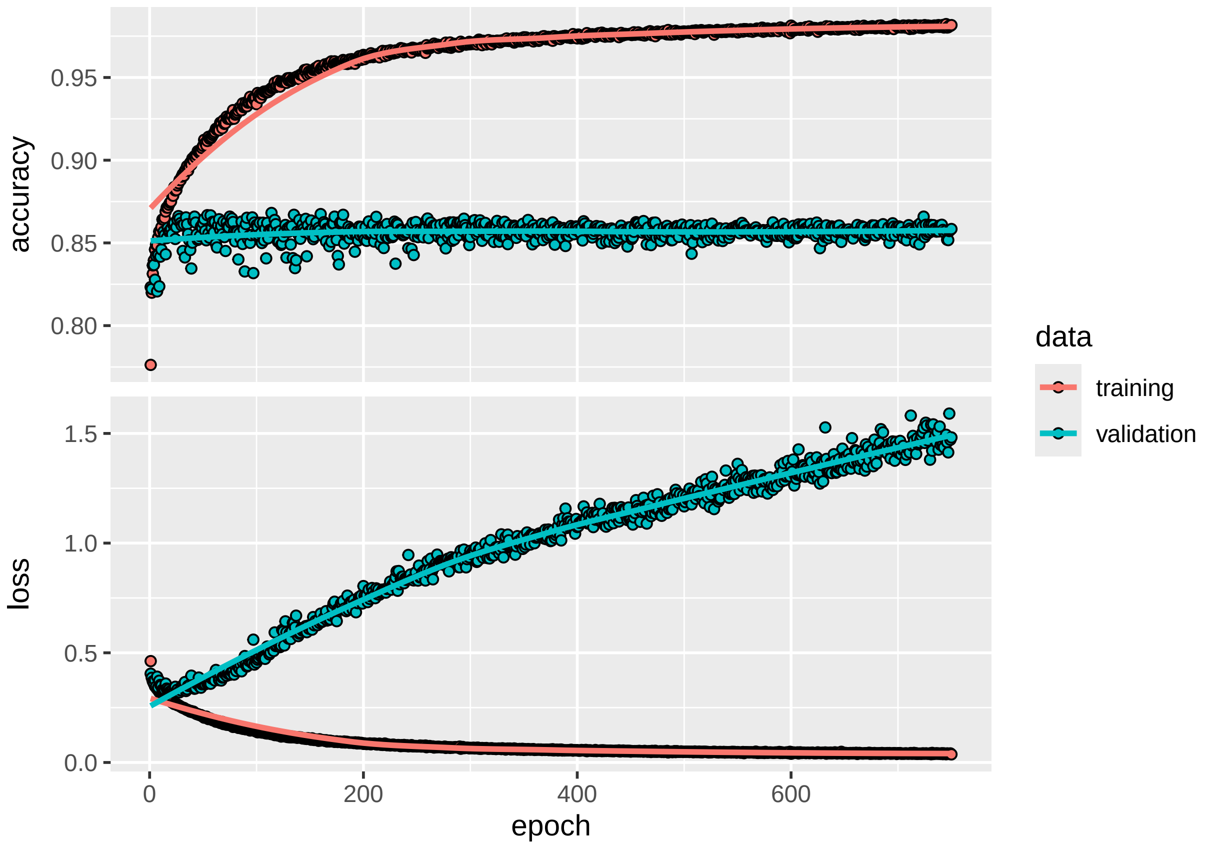
Model 7 - rmsprop, but more units in each hidden layer. Best in terms of overfitting.

```r
set.seed(42)
model7 <- keras_model_sequential() %>%
  layer_dense(units = 348, activation = "relu") %>%
  layer_dense(units = 174, activation = "relu") %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dense(units = 36, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

```r
compile(model7,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")



random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```r
set.seed(42)
history7 <- fit(model7, training_features, training_labels,
               epochs = 750, batch_size = 512, validation_split = 0.33)
```

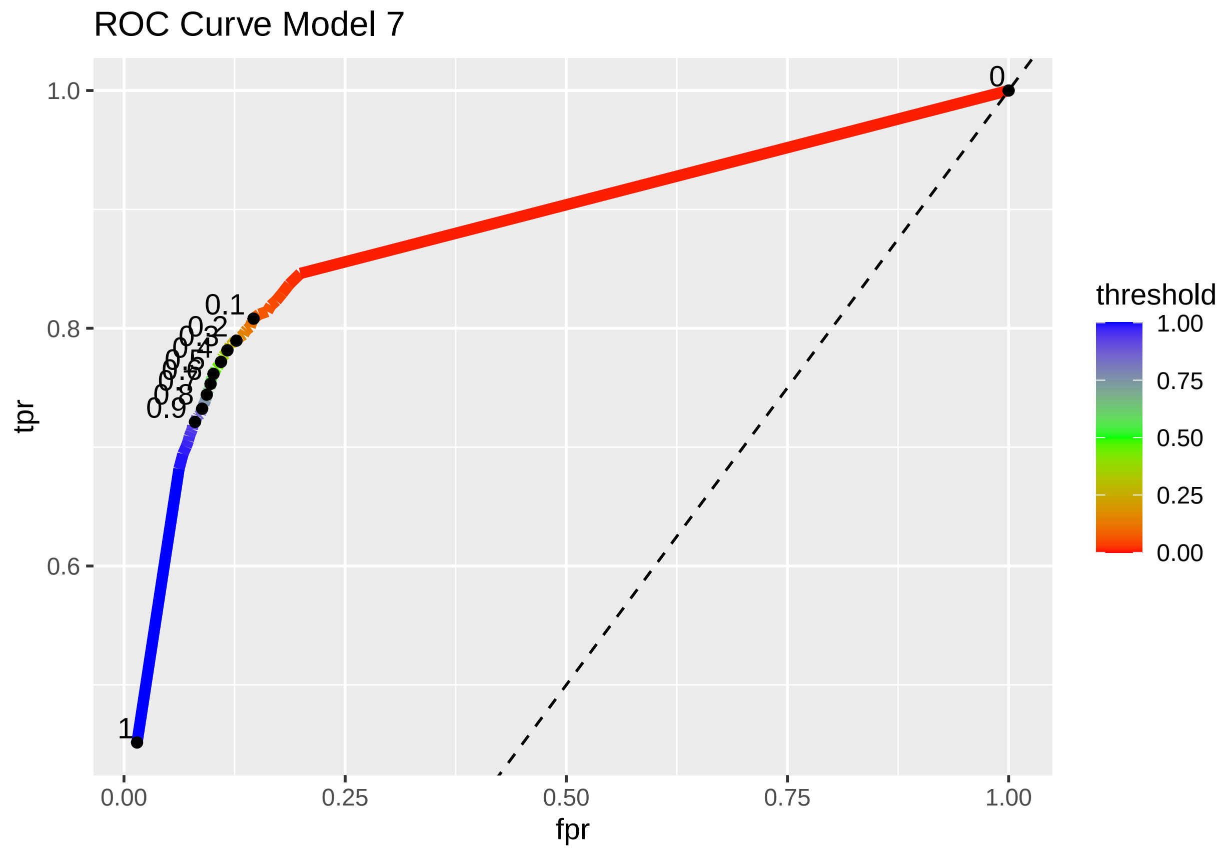
```r
set.seed(42)
plot(history7)
```



```r
predictions7 <- predict(model7, test_features)
```

```
## 284/284 - 1s - 2ms/step
```

```r
test_results7 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions7)
  )

roc_data7 <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data7$threshold) {

  over_threshold <- test_results7[test_results7$p_1 >= i, ]

  fpr <- sum(over_threshold$booking_status==0)/sum(test_results7$booking_status==0)
  roc_data7[roc_data7$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results7$booking_status==1)
  roc_data7[roc_data7$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data7, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data7[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data7[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 7")
```



```
auc7 <- auc(x = roc_data7$fpr, y = roc_data7$tpr, type = "spline")
```

```
## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique
## 'x' values
```

```
auc7
```

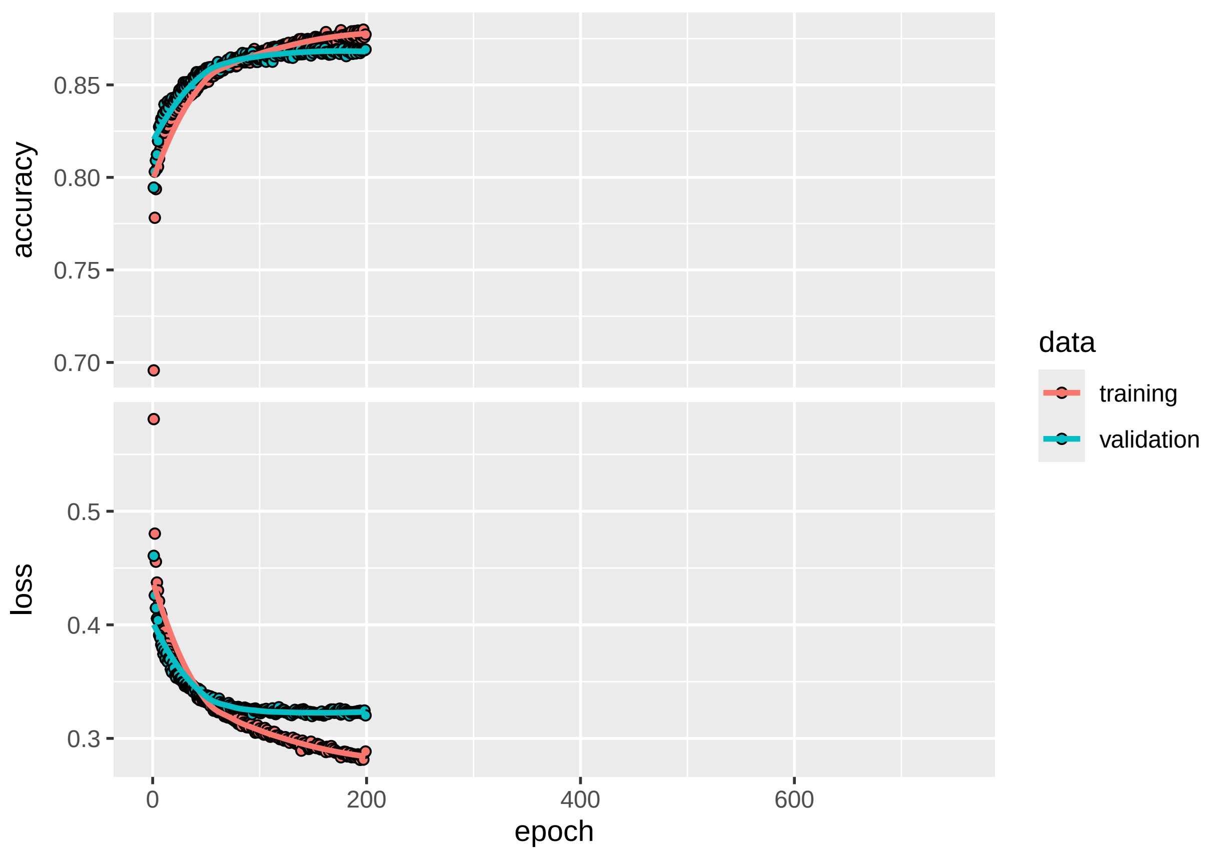
```
## [1] 0.9108024
```

Model 8: Model 7 with dropout to improve generalization

```
set.seed(42)
model8 <- keras_model_sequential() %>%
  layer_dense(units = 348, activation = "relu") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 174, activation = "relu") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 36, activation = "relu") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")


compile(model8,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")



random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```
set.seed(42)
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 50)

history8 <- fit(model8, training_features, training_labels,
               epochs = 750, batch_size = 512, validation_split = 0.33,
               callbacks = list(early_stop))
```

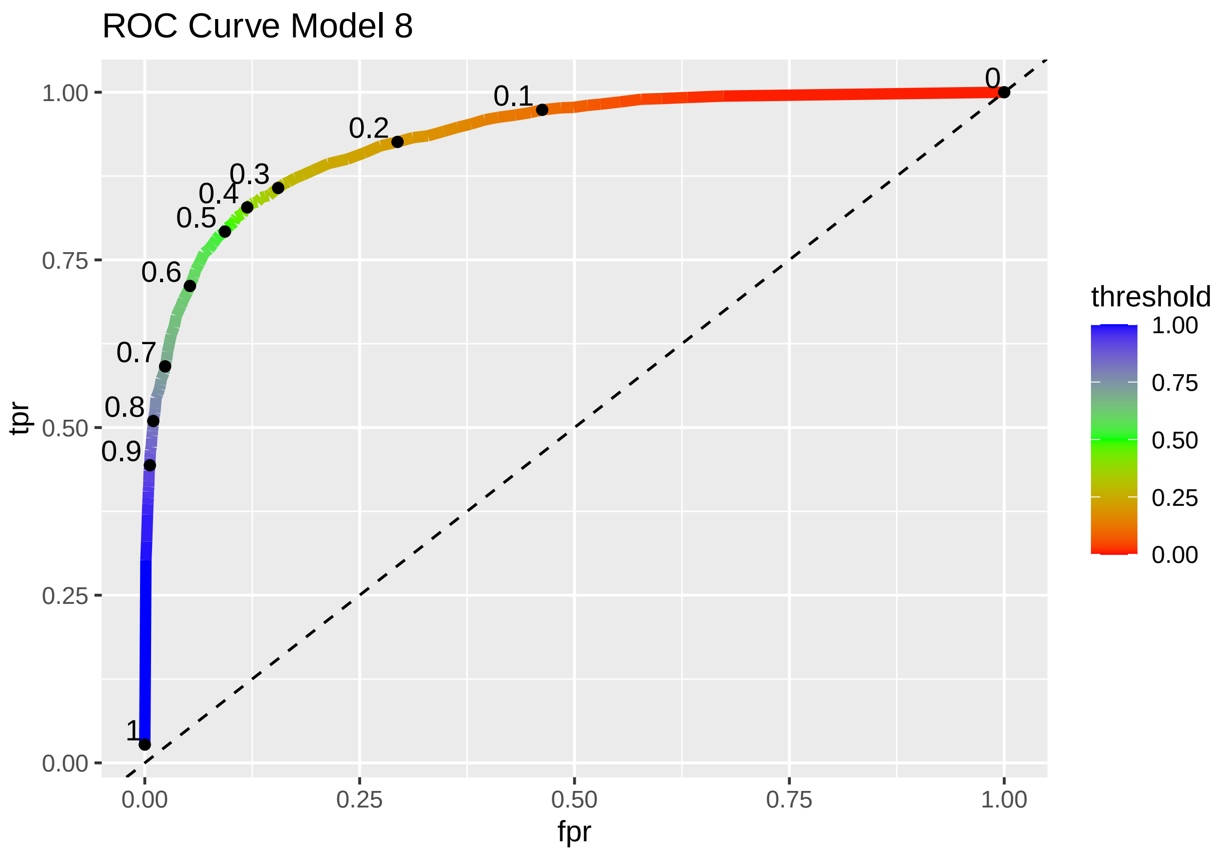
```
set.seed(42)
plot(history8)
```



```r
predictions8 <- predict(model8, test_features)
```

```
## 284/284 - 1s - 2ms/step
```

```r
test_results8 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions8)
  )

roc_data8 <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data8$threshold) {

  over_threshold <- test_results8[test_results8$p_1 >= i, ]

  fpr <- sum(over_threshold$booking_status==0)/sum(test_results8$booking_status==0)
  roc_data8[roc_data8$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results8$booking_status==1)
  roc_data8[roc_data8$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data8, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data8[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data8[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 8")
```



```
auc8 <- auc(x = roc_data8$fpr, y = roc_data8$tpr, type = "spline")
```

```
## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique
## 'x' values
```

```
auc8
```

```
## [1] 0.9315736
```

Preliminary Results (For Reference)

Below are the preliminary results for the binary classification problem where the label is to determine hotel bookings for ABC Hotels that are at a high risk of cancellation. The workflow for creating a densely connected feed-forward neural network for resolving this problem is outlined below.

First, I imported the data, and applied feature engineering to exclude the BookingID to eliminate noise in the dataset, as noted in my Analytic Plan. Next, I converted the positive class (class that I'm interested in determining), which is cancelled bookings to "1", and the non_cancelled class to "0". As noted in my Analytic plan, I was able to extract features from the arrival_date predictor, including both the year and the season. From there, I split the data into a training set (75%) and test set (25%), and created a recipe (booking_recipe) to apply one hot encoding to all categorical variables, center the variables, scale each variable, and prep the data on the training set to use going forward.

Next, the preparation to create a neural network was performed by converting the labels and features on both the training and test sets to arrays. After that, I was able to generate my models. I created five neural network models for this supervised classification problem, and each is outlined below:

Model 1 creation: I considered this model to be my baseline, and specified a model with three layers. I chose three layers because this seemed to be a number where I could appropriately include a sufficient number of units in the model, without adding too much complexity. The first dense layer, or hidden layer, consisted of 87 units, while the second hidden layer consisted of 42 units. I chose these numbers because the total number of features, or predictors, in the dataset was 29 after baking the data. I wanted to be sure to include enough units to capture the interactions and complexities within the data, so I chose three times the amount in the first layer to do this. In the second layer, I chose 42 as this is approximately half of the original 87 units. Finally, the output layer had 1 layer so an overall probability could be generated. Within the hidden layers, I chose RELU (Rectified Linear Unit) as my activation function for a few reasons. First, this activation function introduces non-linearity into the model, allowing to help learn complex patterns within the data. Second, this activation function is effective, yet simple because it involves simple thresholding at 0. For my output layer, I chose sigmoid as my activation function for two main reasons. First, this function outputs a value between 0 and 1, which is ideal because the objective of this problem is to determine a probability between 0 and 1 in which bookings are at high risk for cancellation. Second, sigmoid is ideal for binary classification problems such as this because it ouputs a value between 0 and 1, thus making for simple decision making between two classes.

Compiling Model 1: To compile model 1, and all models, I chose rmsprop as the optimizer. I chose this because rmsprop has an adaptive learning rate for each parameter, which helps in faster convergence and better performance. Additionally, this also helps in dealing with the sparse gradients and ensures more effective updates. I chose binary_crossentropy as my loss function because this particular algorithm measures the performance of a classification model whose output is a probability value between 0 and 1. Since our model uses sigmoid in our output layer, this was a good choice. Also, this loss function is differentiable, making it suitable for gradient-based optimization methods.

Additional model creation: Over the course of the code below, I made singular adjustments from the baseline to each sequential model. In model 2, I used more units in each hidden layer, with 174 in the first layer (six times the amount of predictors) and 87 in the second hidden layer. In model 3, I kept the original amount of units in the hidden layers, but increased the number of epochs from 250 to 750 as a means of iterating through the training set more in an attempt to capture additional data intricacies. In model 4, I increased the number of hidden layers overall as another means of capturing patterns and complex relationships within the data. Finally, in model 5 I combined the increased number of hidden layers with the increased number of epochs to see if the two of these modifications combined would produce better results.

Evaluation of the models: To evaluate the various models, I created plots of each of the history variables to determine the accuracy and loss for the training data, and held out validation data. Looking at each of the accuracy/loss plots, a few general conclusions can be made. First, none of these models appear to underfit to the data. I can tell this is the case because underfitting typically comes with the appearance of at least a few of the following visual indicators: 1) High training loss curve 2) Low training accuracy 3) Similar training and validation performance. None of these are occurring. However, there is some overfitting happening. Typical signs of overfitting on the various training and validation plots include the following: 1) Low training loss with high validation loss 2) High training accuracy with low validation accuracy 3) Diverging training and validation curves. In the models I've created below, overfitting occurs and is illustrated by reasons 1) and 3) above. Model 5 seems to overfit the most, which is to be expected because it is the most complex model due to the highest number of epochs and highest number of hidden layers.

Next Steps: In the next phase of the project, I would like to dial in the various models below to find the optimal, or champion, result. This will provide me with the best means of predicting whether a booking is of high risk for cancellation for ABC Hotels. Achieving overfitting is a good thing, because now I can apply methods to reduce overfitting, while working to effectively generalize on new data (test set). Some of the methods I would like to use to do this in the final project phase include earlier stopping on the training data, simplification of the final model, or application of L1/L2 regularization. As an additional quick means of validation, I also generated ROC/AUC data on the held out test set. Model 2 produced the highest AUC (area under the curve) score, so it may be beneficial to take a closer look at the qualities that helped this model perform so well.

```r
library(tidymodels)
library(reticulate)
library(tensorflow)
library(keras3)
library(MESS)

getwd()
```

```
## [1] "C:/Users/steve/OneDrive/Documents/School/DSE6211/Project"
```

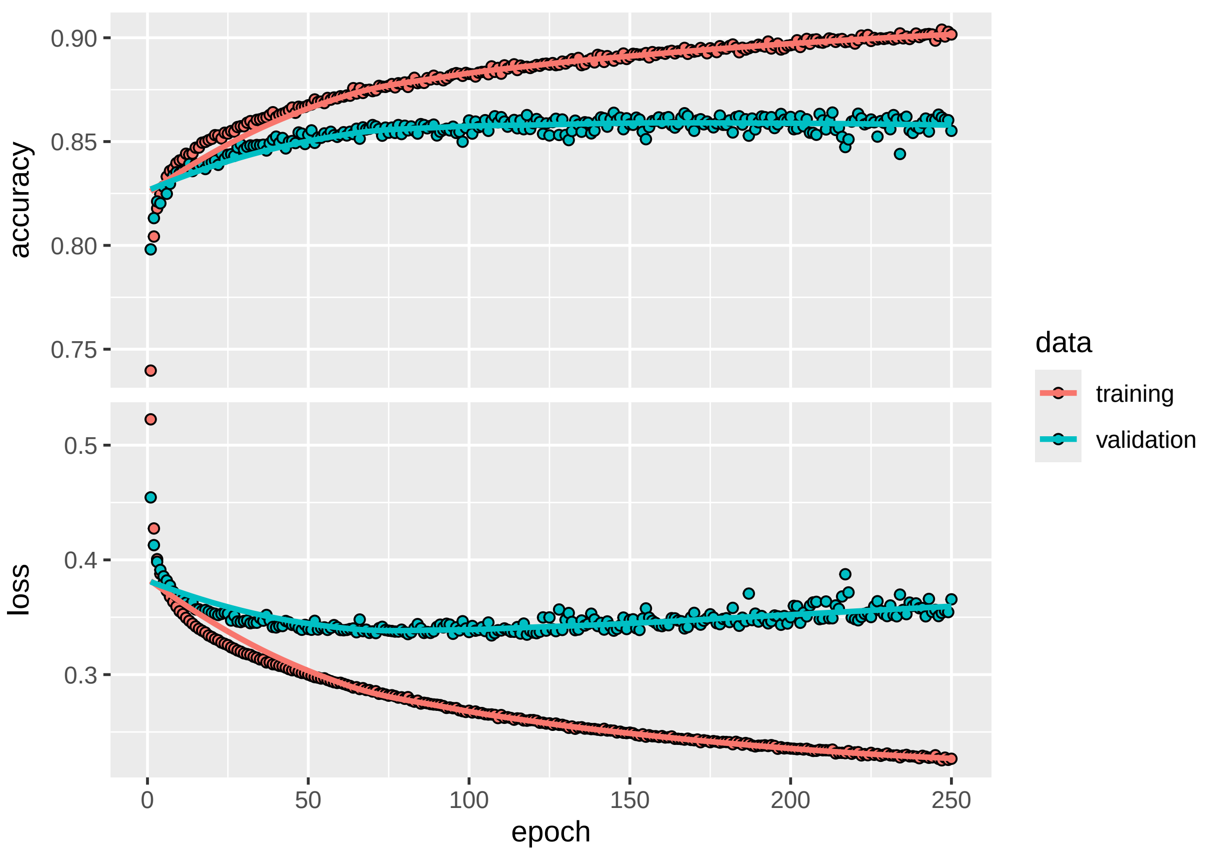
```r
setwd("C:\\Users\\steve\\OneDrive\\Documents\\School\\DSE6211\\Project")

data <- read.csv("project_data.csv")

data <- data[, -1]
data$booking_status <- ifelse(data$booking_status == "canceled", 1, 0)

#Function creation to extract seasons from arrival date
data$arrival_date <- as.Date(data$arrival_date)
data$month <- format(data$arrival_date, "%m")
data$year <- format(data$arrival_date, "%Y")

get_season <- function(month) {
  month <- as.numeric(month)
  if (month %in% c(12, 1, 2)) {
    return("Winter")
  } else if (month %in% c(3, 4, 5)) {
    return("Spring")
  } else if (month %in% c(6, 7, 8)) {
    return("Summer")
  } else if (month %in% c(9, 10, 11)) {
    return("Autumn")
  }
}

data$season <- sapply(data$month, get_season)

data <- data[, -9]
data <- data[, -16]

set.seed(42)
data_split <- initial_split(data, strata = "booking_status", prop = 0.75)

training_set <- training(data_split)
test_set  <- testing(data_split)

booking_recipe <-
  recipe(
    booking_status ~ .,
    data = training_set
  ) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  prep(training = training_set, retain = TRUE)

training_set_baked <- bake(booking_recipe, new_data = training_set)
test_set_baked <- bake(booking_recipe, new_data = test_set)



training_features <- array(data = unlist(training_set_baked[, -12]),
                           dim = c(nrow(training_set_baked), 28))
training_labels <- array(data = unlist(training_set_baked[, 12]),
                         dim = c(nrow(training_set_baked)))

test_features <- array(data = unlist(test_set_baked[, -12]),
                       dim = c(nrow(test_set_baked), 28))
test_labels <- array(data = unlist(test_set_baked[, 12]),
                     dim = c(nrow(test_set_baked)))

use_virtualenv("my_tf_workspace")
```

Model 1 - Baseline model

```r
set.seed(42)
model1 <- keras_model_sequential() %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dense(units = 42, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

set.seed(42)
compile(model1,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")

set.seed(42)
random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```r
set.seed(42)
history1 <- fit(model1, training_features, training_labels,
               epochs = 250, batch_size = 512, validation_split = 0.33)
```

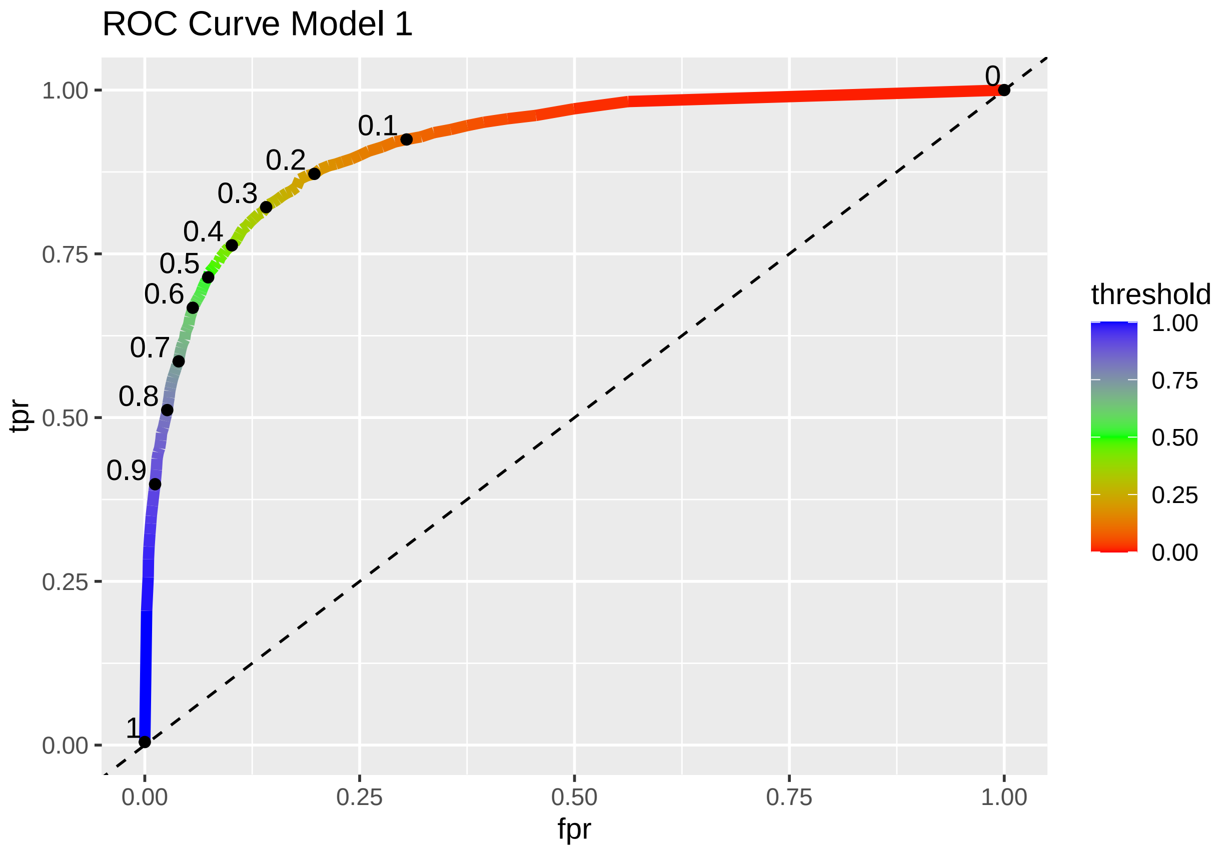
```r
plot(history1)
```



```
set.seed(42)
predictions1 <- predict(model1, test_features)
```

```
## 284/284 - 0s - 2ms/step
```

```
test_results1 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions1)
  )

roc_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data$threshold) {

  over_threshold <- test_results1[test_results1$p_1 >= i, ]

  fpr <- sum(over_threshold$booking_status==0)/sum(test_results1$booking_status==0)
  roc_data[roc_data$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results1$booking_status==1)
  roc_data[roc_data$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 1")
```



```
auc1 <- auc(x = roc_data$fpr, y = roc_data$tpr, type = "spline")
auc1
```

```
## [1] 0.9193438
```

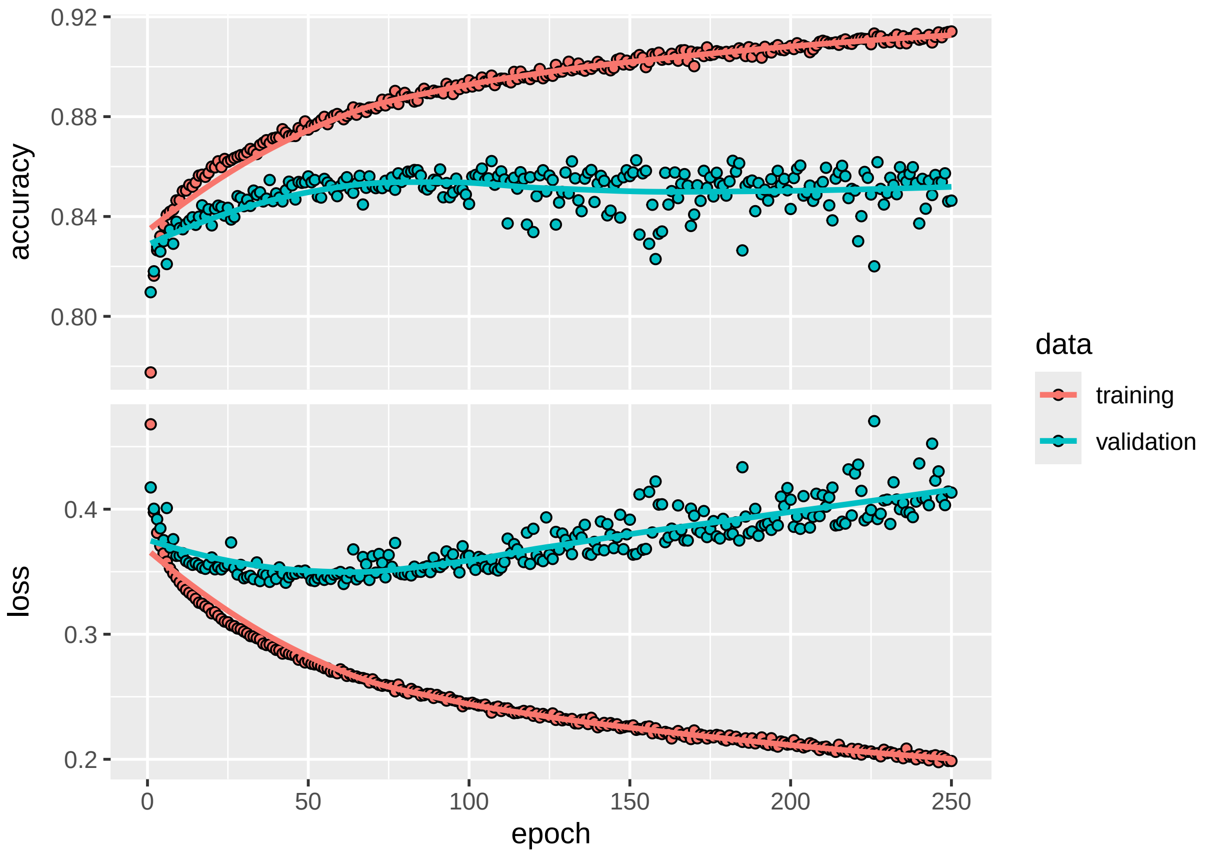
Model 2 - Additional units in hidden layer

```
set.seed(42)
model2 <- keras_model_sequential() %>%
  layer_dense(units = 174, activation = "relu") %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

set.seed(42)
compile(model2,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")
```

```r
set.seed(42)
random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```r
set.seed(42)
history2 <- fit(model2, training_features, training_labels,
               epochs = 250, batch_size = 512, validation_split = 0.33)
```

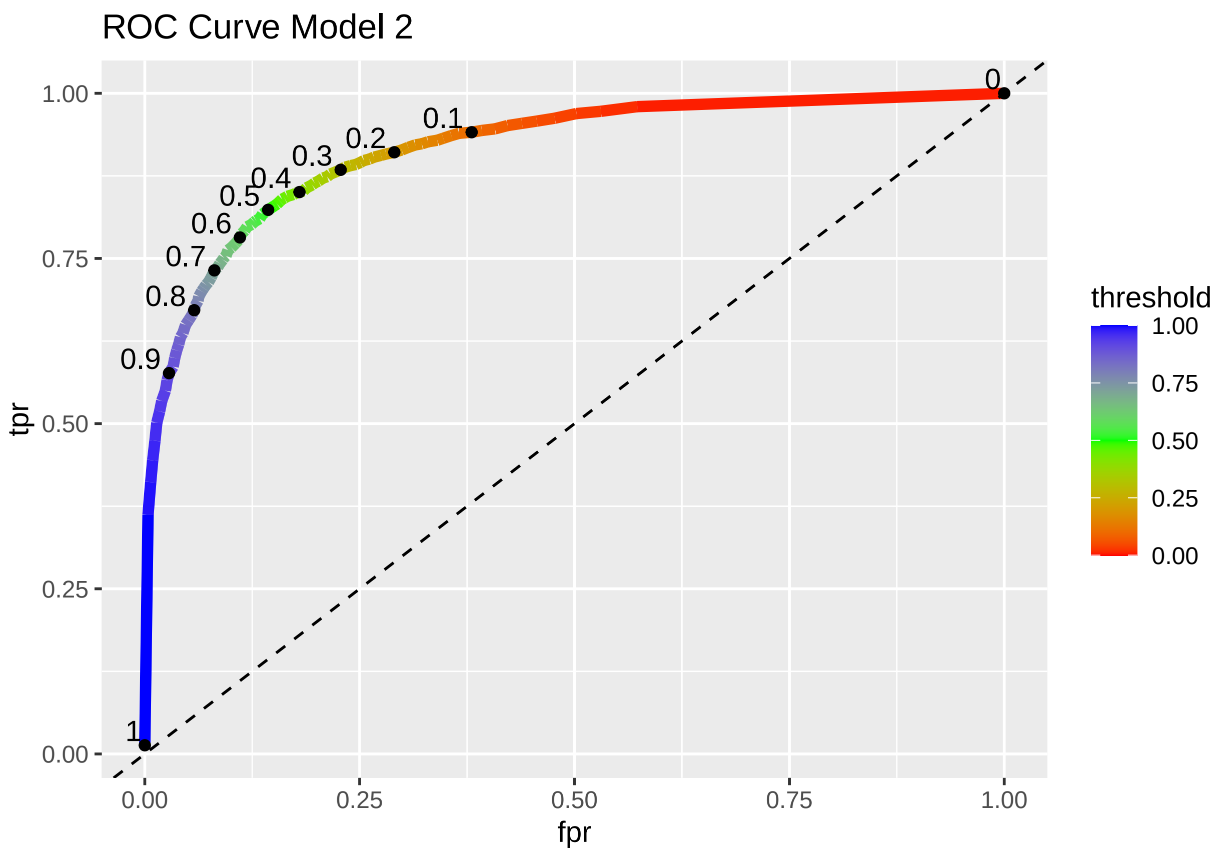
```r
plot(history2)
```



```r
set.seed(42)
predictions2 <- predict(model2, test_features)
```

```
## 284/284 - 0s - 631us/step
```

```r
test_results2 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions2)
  )

roc_data2 <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data2$threshold) {

  over_threshold <- test_results2[test_results2$p_1 >= i, ]
  fpr <- sum(over_threshold$booking_status==0)/sum(test_results2$booking_status==0)
  roc_data2[roc_data2$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results2$booking_status==1)
  roc_data2[roc_data2$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data2, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data2[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data2[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 2")
```



```
auc2 <- auc(x = roc_data2$fpr, y = roc_data2$tpr, type = "spline")
auc2
```

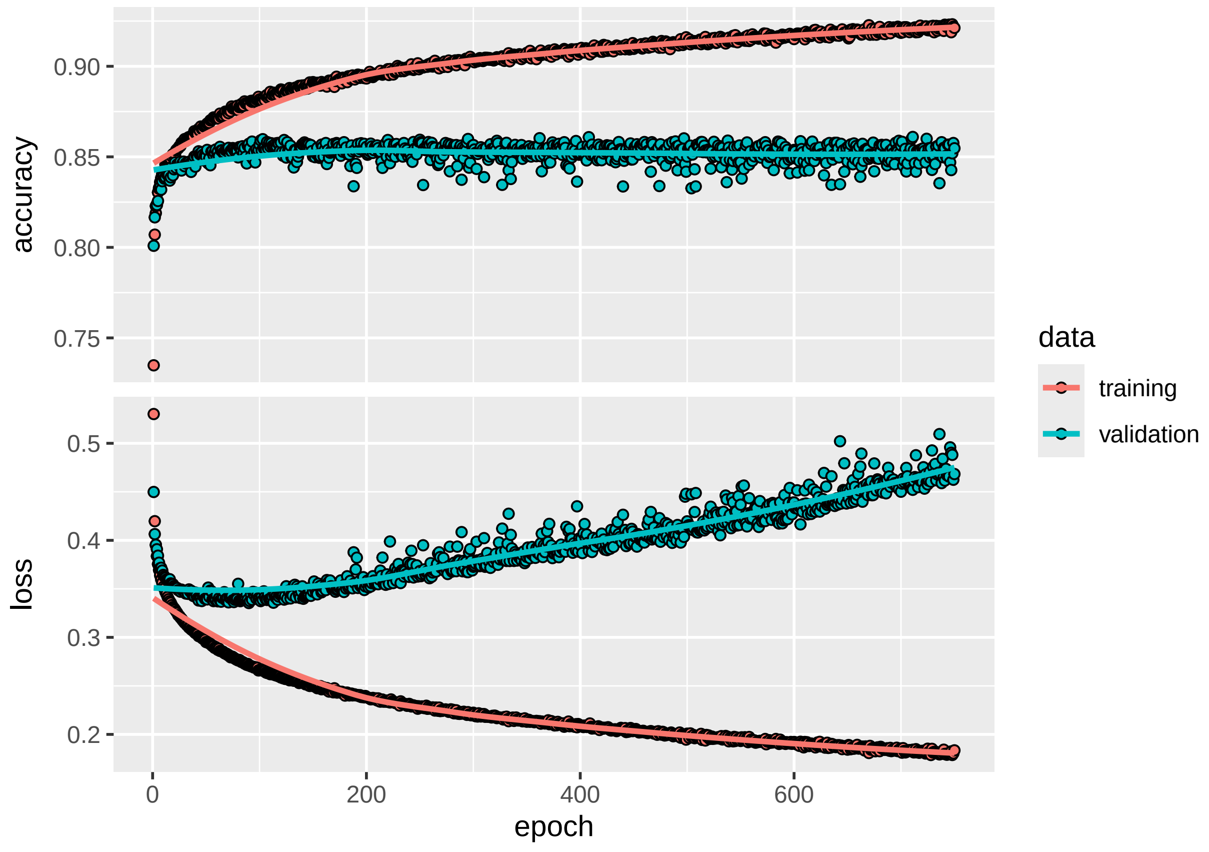
```
## [1] 0.9198158
```

#Model 3 - More epochs, same units in hidden layer as original

```
set.seed(42)
model3 <- keras_model_sequential() %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dense(units = 42, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

set.seed(42)
compile(model3,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")


set.seed(42)
random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```
set.seed(42)
history3 <- fit(model3, training_features, training_labels,
               epochs = 750, batch_size = 512, validation_split = 0.33)
```

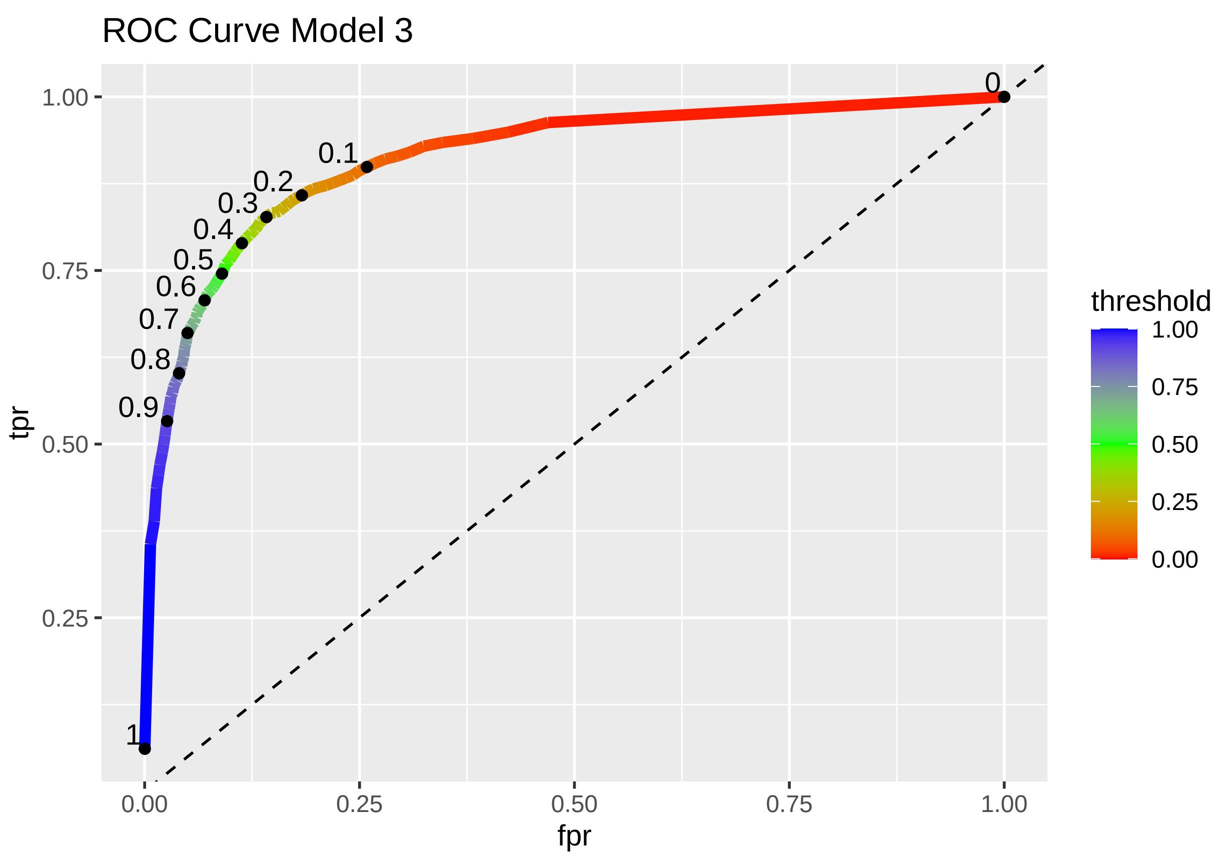
```
plot(history3)
```



```
set.seed(42)
predictions3 <- predict(model3, test_features)
```

```
## 284/284 - 0s - 558us/step
```

```r
test_results3 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions3)
  )

roc_data3 <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data3$threshold) {

  over_threshold <- test_results3[test_results3$p_1 >= i, ]

  fpr <- sum(over_threshold$booking_status==0)/sum(test_results3$booking_status==0)
  roc_data3[roc_data3$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results3$booking_status==1)
  roc_data3[roc_data3$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data3, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data3[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data3[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 3")
```



```r
auc3 <- auc(x = roc_data3$fpr, y = roc_data3$tpr, type = "spline")
auc3
```

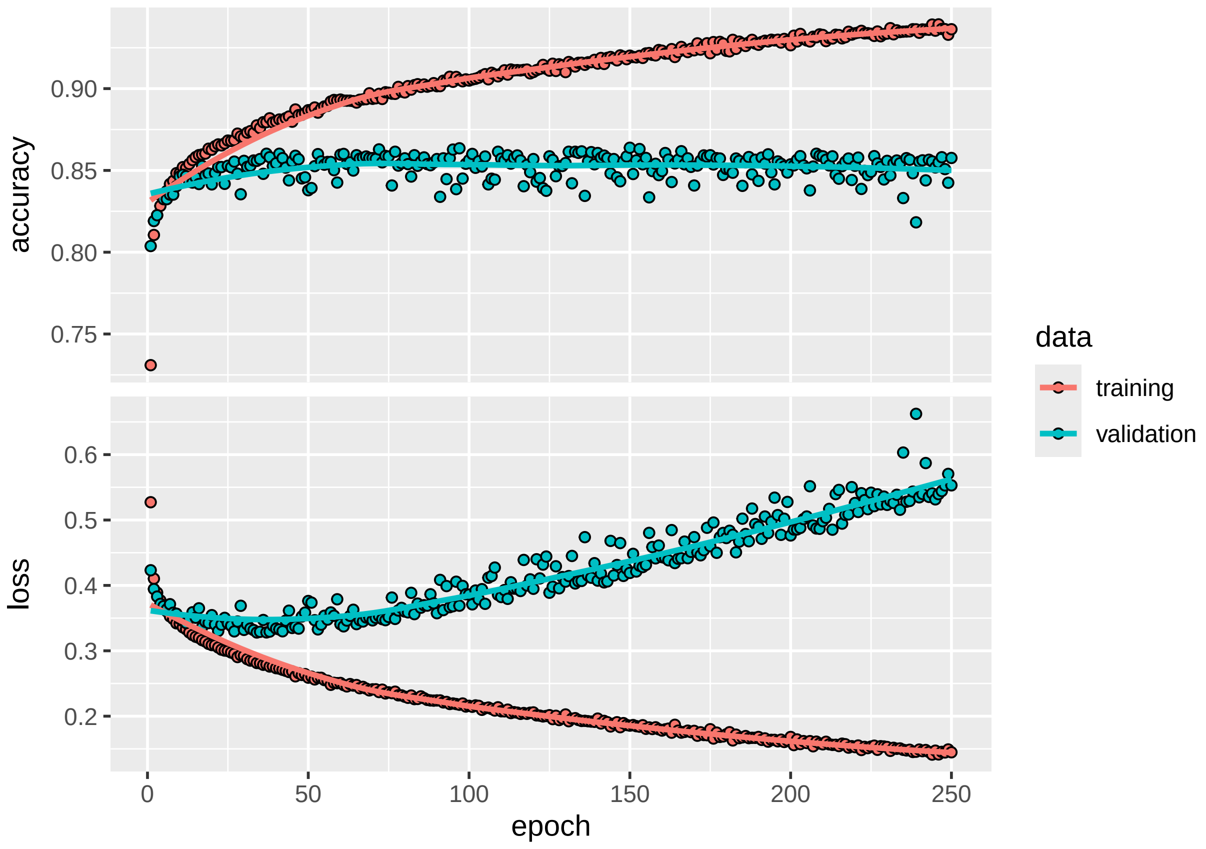
```
## [1] 0.9206474
```

#Model 4 - Add more layers, 250 epochs, same # of units in hidden layer as original

```r
set.seed(42)
model4 <- keras_model_sequential() %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dense(units = 70, activation = "relu") %>%
  layer_dense(units = 53, activation = "relu") %>%
  layer_dense(units = 36, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

set.seed(42)
compile(model4,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")

set.seed(42)
random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```r
set.seed(42)
history4 <- fit(model4, training_features, training_labels,
                epochs = 250, batch_size = 512, validation_split = 0.33)
```

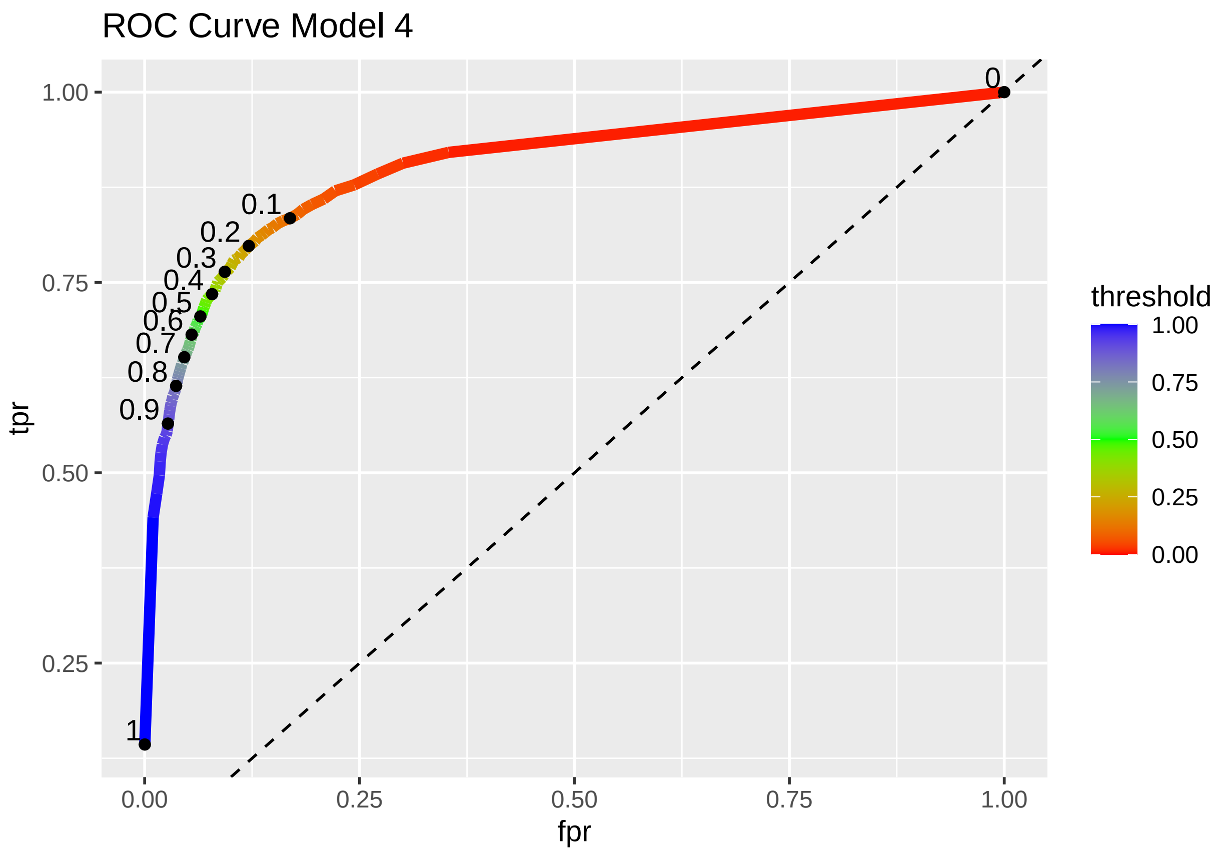
```
plot(history4)
```



```
set.seed(42)
predictions4 <- predict(model4, test_features)
```

```
## 284/284 - 0s - 673us/step
```

```
test_results4 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions4)
  )

roc_data4 <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data4$threshold) {

  over_threshold <- test_results4[test_results4$p_1 >= i, ]

  fpr <- sum(over_threshold$booking_status==0)/sum(test_results4$booking_status==0)
  roc_data4[roc_data4$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results4$booking_status==1)
  roc_data4[roc_data4$threshold==i, "tpr"] <- tpr
}

ggplot() +
  geom_line(data = roc_data4, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data4[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data4[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 4")
```



```
auc4 <- auc(x = roc_data4$fpr, y = roc_data4$tpr, type = "spline")
auc4
```

```
## [1] 0.9065901
```

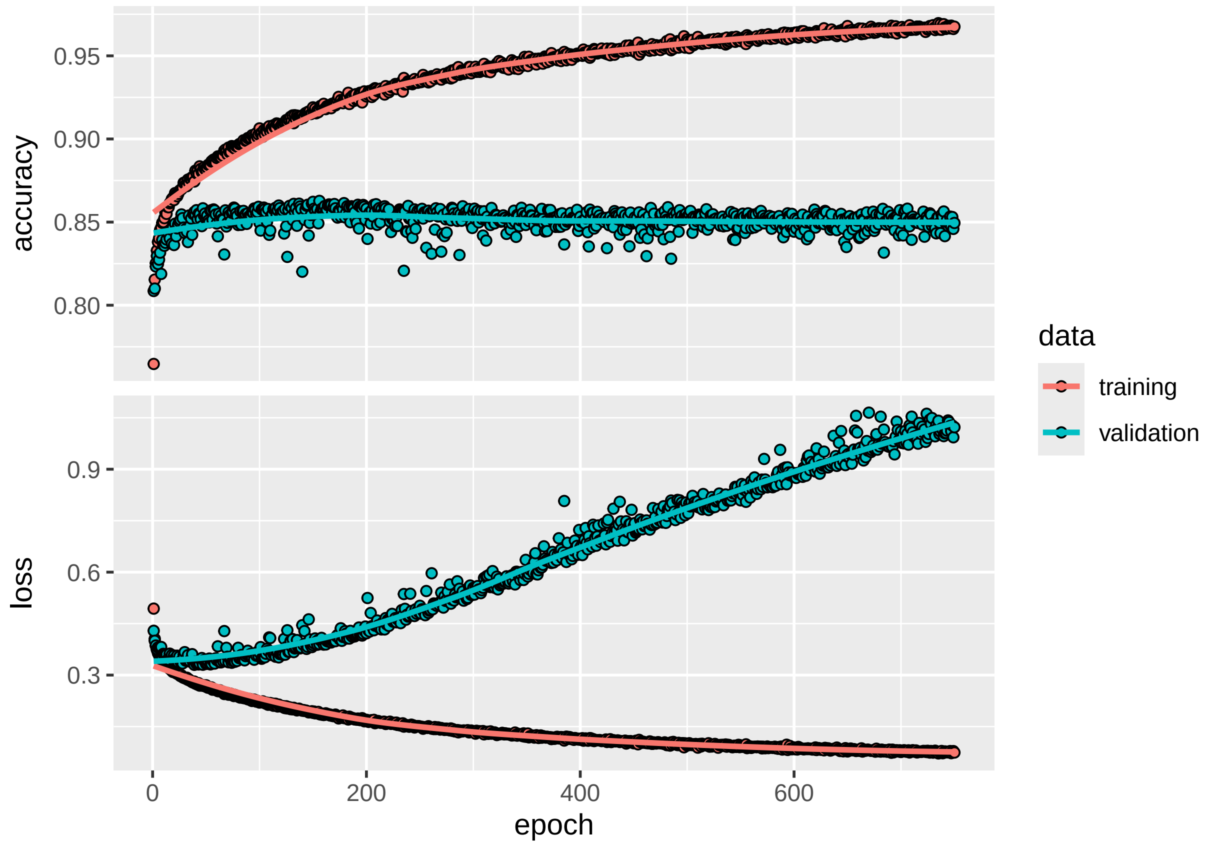
#Model 5 - Combining the two most meaningful models. More hidden layers, and 750 epochs. The goal is to overfit

```
set.seed(42)
model5 <- keras_model_sequential() %>%
  layer_dense(units = 87, activation = "relu") %>%
  layer_dense(units = 70, activation = "relu") %>%
  layer_dense(units = 53, activation = "relu") %>%
  layer_dense(units = 36, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

```r
set.seed(42)
compile(model5,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")


set.seed(42)
random_shuffle <- sample(1:nrow(training_features))
training_features <- training_features[random_shuffle, ]
training_labels <- training_labels[random_shuffle]
```

```r
set.seed(42)
history5 <- fit(model5, training_features, training_labels,
               epochs = 750, batch_size = 512, validation_split = 0.33)
```

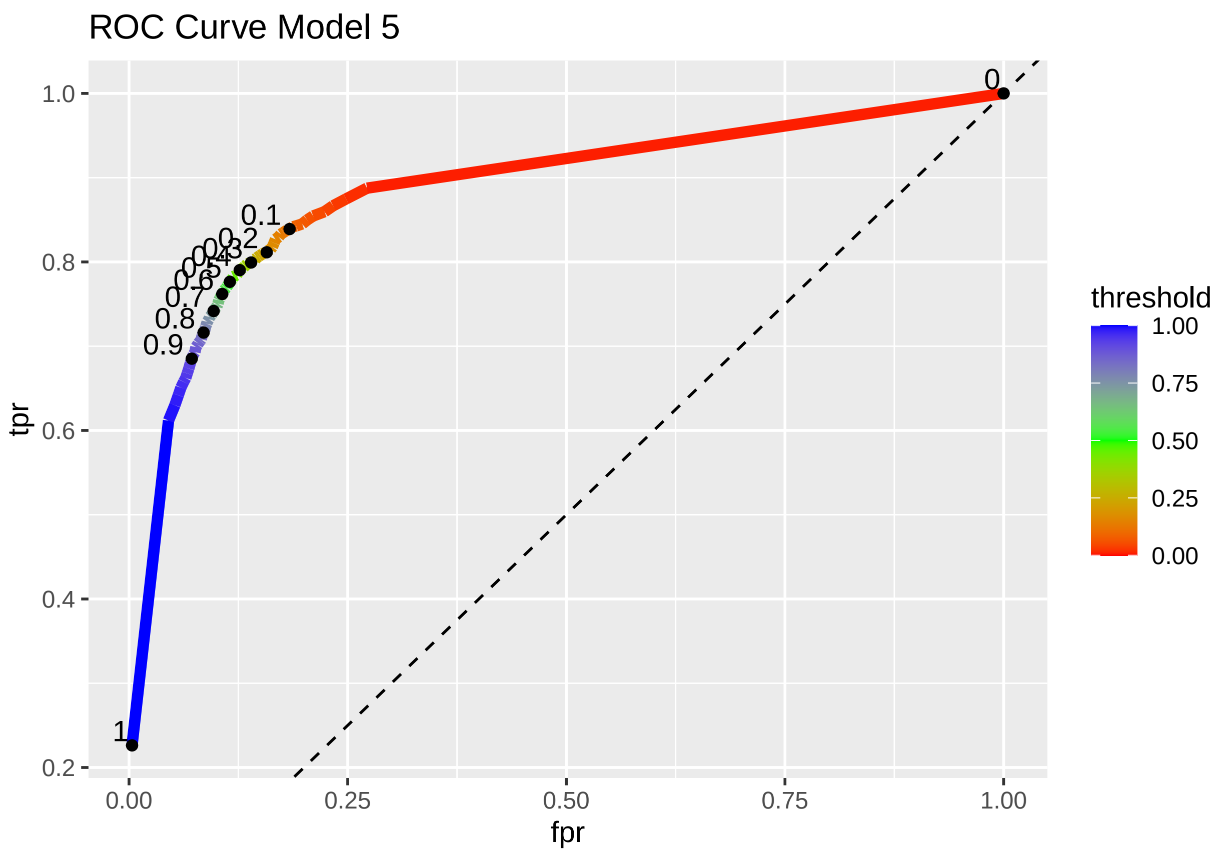
```r
plot(history5)
```



```r
set.seed(42)
predictions5 <- predict(model5, test_features)
```

```
## 284/284 - 0s - 699us/step
```

```r
test_results5 <-
  test_set %>%
  select(booking_status) %>%
  bind_cols(
    data.frame(p_1 = predictions5)
  )

roc_data5 <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data5$threshold) {

  over_threshold <- test_results5[test_results5$p_1 >= i, ]

  fpr <- sum(over_threshold$booking_status==0)/sum(test_results5$booking_status==0)
  roc_data5[roc_data5$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$booking_status==1)/sum(test_results5$booking_status==1)
  roc_data5[roc_data5$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data5, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data5[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data5[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2)) +
  labs(title = "ROC Curve Model 5")
```



```r
auc5 <- auc(x = roc_data5$fpr, y = roc_data5$tpr, type = "spline")
auc5
```

```
## [1] 0.9124911
```