Assignment 1

Jacob Fabian

2023-02-13

##First Model

```
library(keras)
imdb <- dataset_imdb(num_words =10000)
c(c(train_data, train_labels), c(test_data, test_labels)) %<-% imdb
```

train_labels and test_labels are lists of 0s and 1s where 0 stands for negative and 1 stands for positive

```
##  int [1:218] 1 14 22 16 43 530 973 1622 1385 65 ...
```

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

From the first entry, the train label results a 1 and indicates a positive review for the first entry.

#Prepare the Data

```
vectorize_sequences <- function(sequences, dimension = 10000) {
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for(i in 1:length(sequences))
    results[i, sequences[[i]]] <- 1
  results
}
```

#Vectorize our train and test data.

```
x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)
str(x_train[1,])
```

```
##  num [1:10000] 1 1 0 1 1 1 1 1 1 0 ...
```

The first two entries in the train data are classified as positive and the third is negative.

#Building

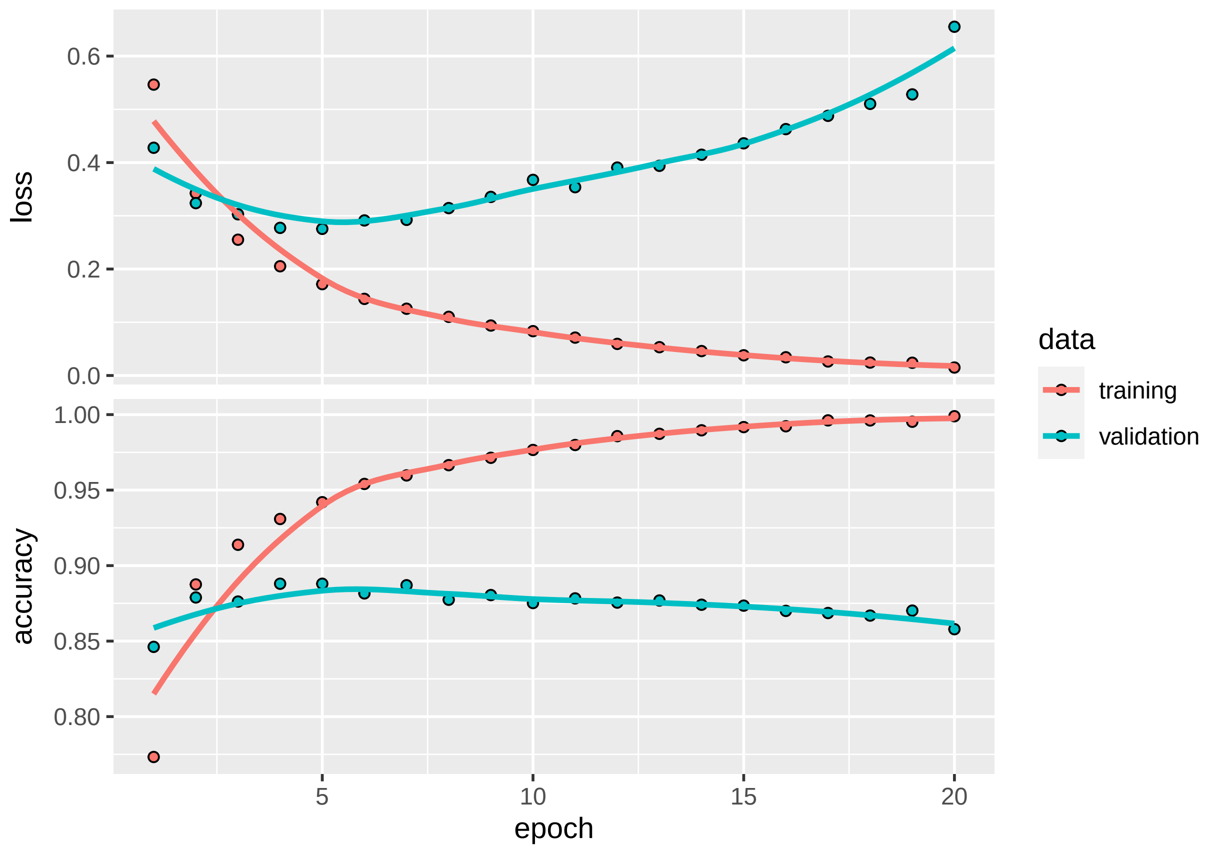
```
library(keras)
model <- keras_model_sequential() %>% layer_dense(units = 16, activation = "relu", input_shape = c(1000
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

#Validating

```
val_indices <- 1:10000
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

#Training

```
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
plot(history)
```



#Train another model to avoid overfitting and using less epochs.

```
model <- keras_model_sequential() %>% layer_dense(units = 16, activation = "relu", input_shape = c(1000
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
model %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
##      loss  accuracy
## 0.2966069 0.8823200
```

#Predict model

```
model %>% predict(x_test[1:10,])
```

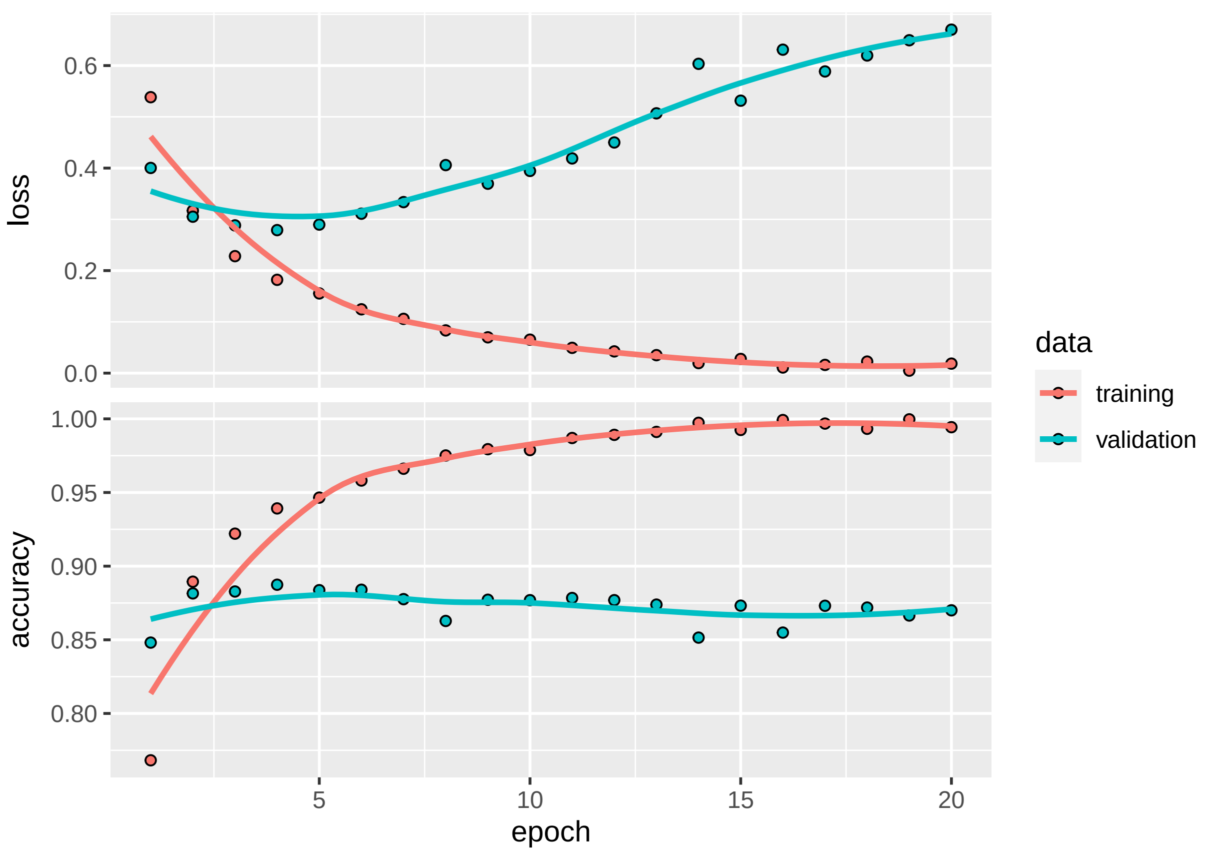
```
##              [,1]
##  [1,] 0.23485234
##  [2,] 0.99941903
##  [3,] 0.94013399
##  [4,] 0.90914708
##  [5,] 0.96296000
##  [6,] 0.83963221
##  [7,] 0.99972850
##  [8,] 0.01516275
##  [9,] 0.97444600
## [10,] 0.99673414
```

#Question 2 Changing the amount of layers

```
library(keras)
model_2 <- keras_model_sequential() %>% layer_dense(units = 16, activation = "relu", input_shape = c(10
model_2 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

```
val_indices <- 1:10000
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

```
history2 <- model_2 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
plot(history2)
```



```
model_2 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- model_2 %>% evaluate(x_test, y_test)
results
```

```
##      loss  accuracy
## 0.4876051 0.8615600
```

```
model_2 %>% predict(x_test[1:10,])
```

```
##               [,1]
##  [1,] 0.020021237
##  [2,] 0.999999344
##  [3,] 0.083478399
##  [4,] 0.820506215
##  [5,] 0.996648312
##  [6,] 0.997727573
##  [7,] 0.999996245
##  [8,] 0.001356122
##  [9,] 0.992386281
## [10,] 0.999993265
```

After comparing the prediction to the original, we can see that when we add a another layer, we have a higher loss than previously and lose accuracy.

Comparing the two predictions, the model was confident and became more confident, the units with low confidence became even less confident

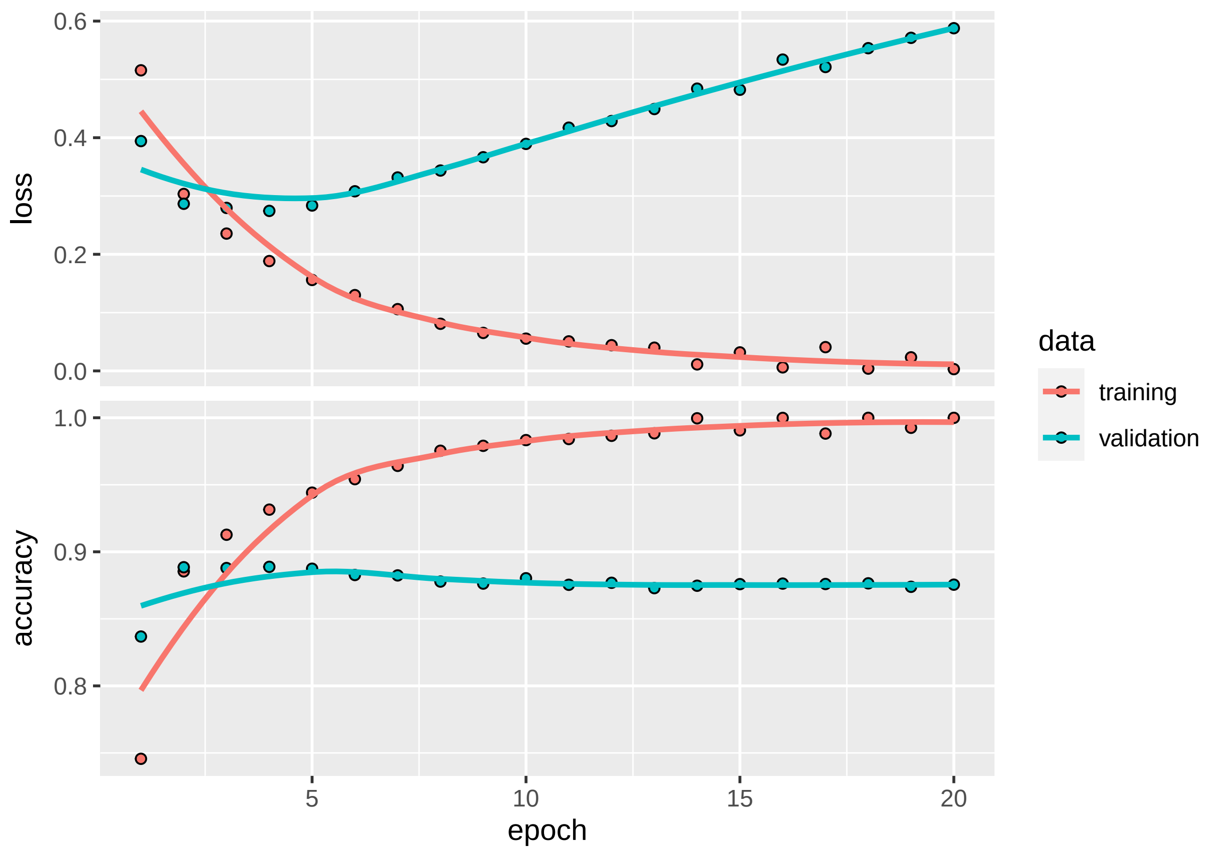
I believe this is caused by overfitting of the training data.

#Changing Layer Units

```
library(keras)
model_3 <- keras_model_sequential() %>% layer_dense(units = 64, activation = "relu", input_shape = c(10
model_3 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

```
val_indices <- 1:10000
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

```
history3 <- model_3 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
plot(history3)
```



```
model_3 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- model_3 %>% evaluate(x_test, y_test)
results
```

```
##      loss  accuracy
## 0.4810466 0.8594800
```

```
model_3 %>% predict(x_test[1:10,])
```

```
##               [,1]
##  [1,] 0.0400041491
##  [2,] 0.9999999404
##  [3,] 0.9888625145
##  [4,] 0.9950872660
##  [5,] 0.9983879328
##  [6,] 0.9996635318
##  [7,] 0.9992673397
##  [8,] 0.0002565149
##  [9,] 0.9983392358
## [10,] 0.9999875426
```

Adding units had an impact on accuracy and loss.

This change doubled the loss from previously and decreased accuracy by .02

Again we see the model has a reactions to what it believes is correct and incorrect. Even comparing it to the first one, this model is 100% confident in [2,],[7,], and [10,]. Yet, it is less confident than before for [1,] and [3,].

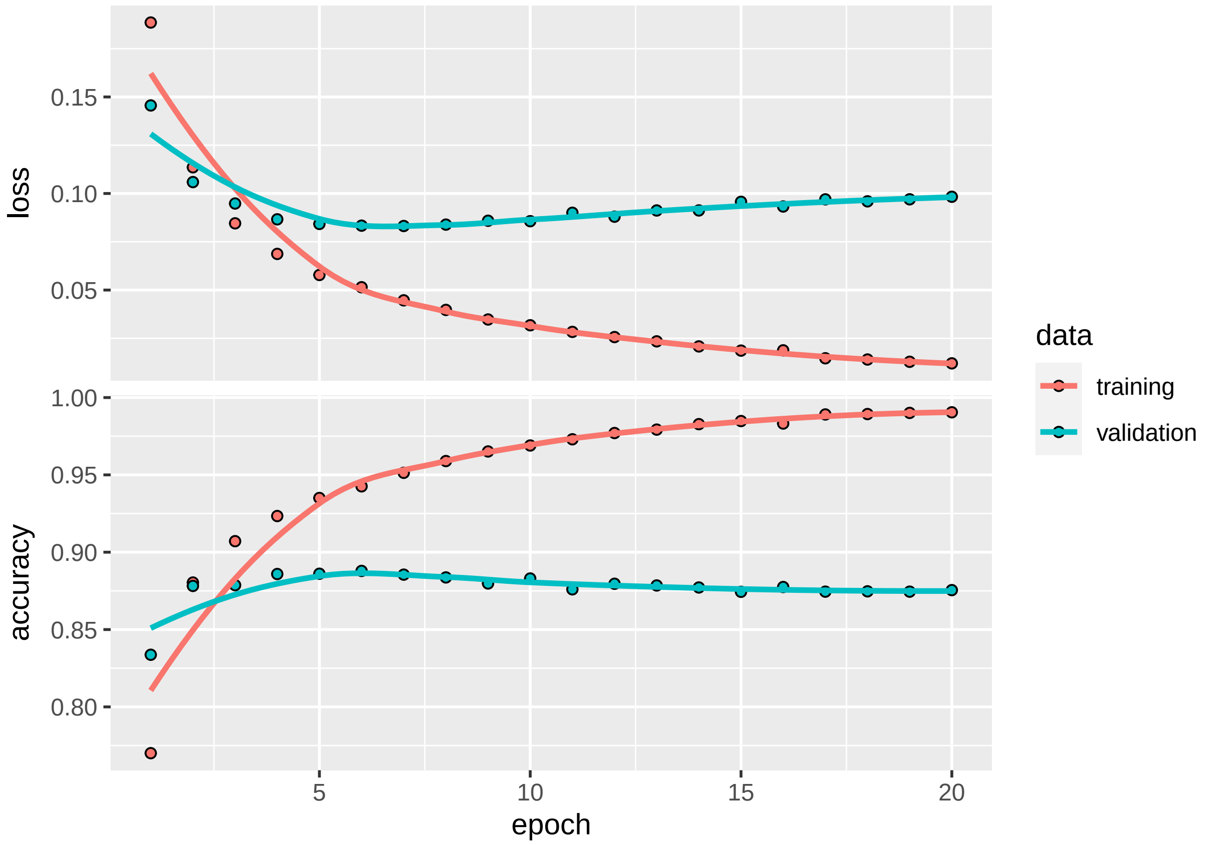
Both of these observations, adding to these models in both units and layers makes the model more overfit with the train data and preforms poorly on the validation and test sets.

Question 3 - MSE

```
library(keras)
model_4 <- keras_model_sequential() %>% layer_dense(units = 16, activation = "relu", input_shape = c(10
model_4 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
```

```
val_indices <- 1:10000
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

```
history4 <- model_4 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
plot(history4)
```



```
model_4 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- model_4 %>% evaluate(x_test, y_test)
results
```

```
##      loss  accuracy
## 0.1030864 0.8692800
```

```
model_4 %>% predict(x_test[1:10,])
```

```
##                 [,1]
##  [1,] 0.0235024244
##  [2,] 0.9999865890
##  [3,] 0.4386935830
##  [4,] 0.8919990659
##  [5,] 0.9951653481
##  [6,] 0.9828797579
##  [7,] 0.9995523095
##  [8,] 0.0007448394
##  [9,] 0.9848963618
## [10,] 0.9998021126
```

By changing the loss function, we decrease the loss at the expense of a loss in accuracy.

The model decreases confidence to how the other adjustments have, and [9,] even loses some confidence.

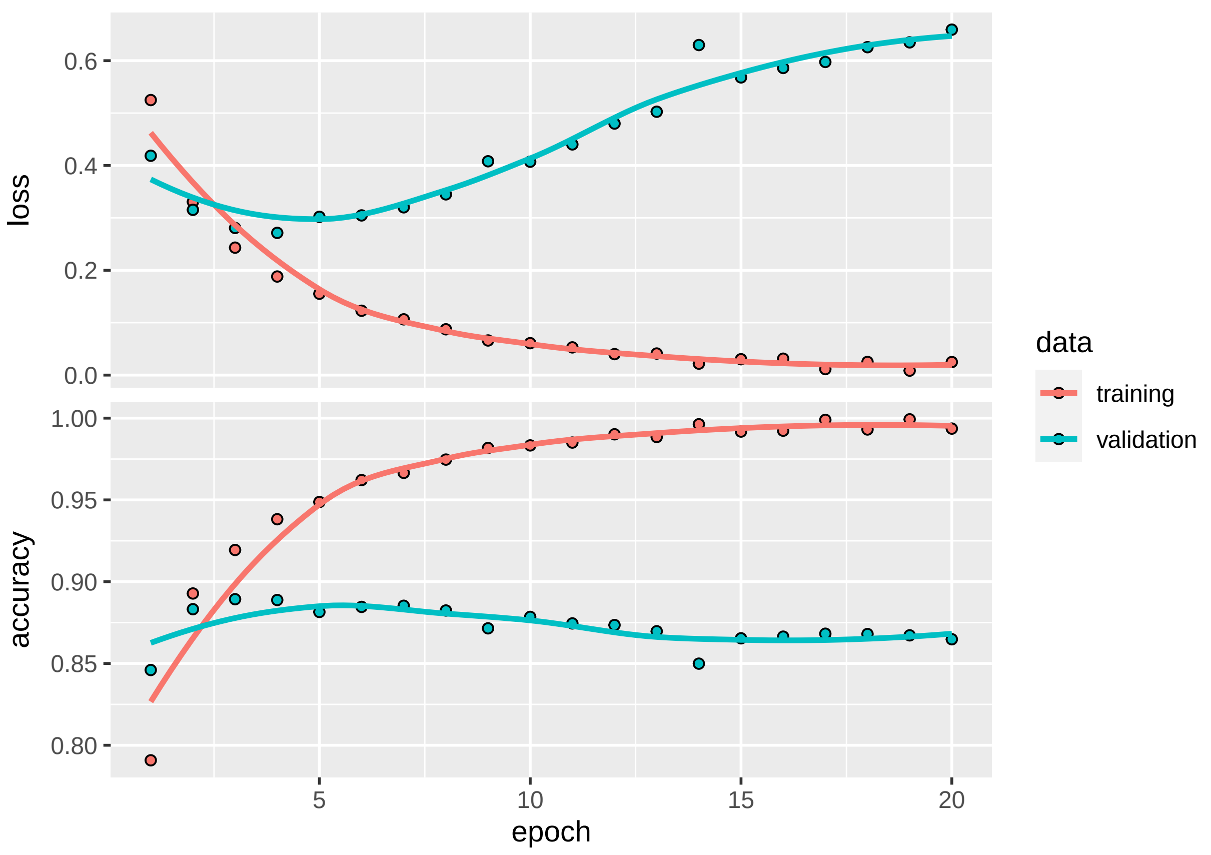
This is due to how the loss functions measures loss. MSE is Mean Squared Error which is less applicable in this case since the output is forced into the binary form.

Question 4 - TSNH

```
library(keras)
model_5 <- keras_model_sequential() %>% layer_dense(units = 16, activation = "tanh", input_shape = c(10
model_5 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

```
val_indices <- 1:10000
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

```
history5 <- model_5 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
plot(history5)
```



```
model_5 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- model_5 %>% evaluate(x_test, y_test)
results
```

```
##      loss  accuracy
## 0.4571963 0.8607600
```

```
model_5 %>% predict(x_test[1:10,])
```

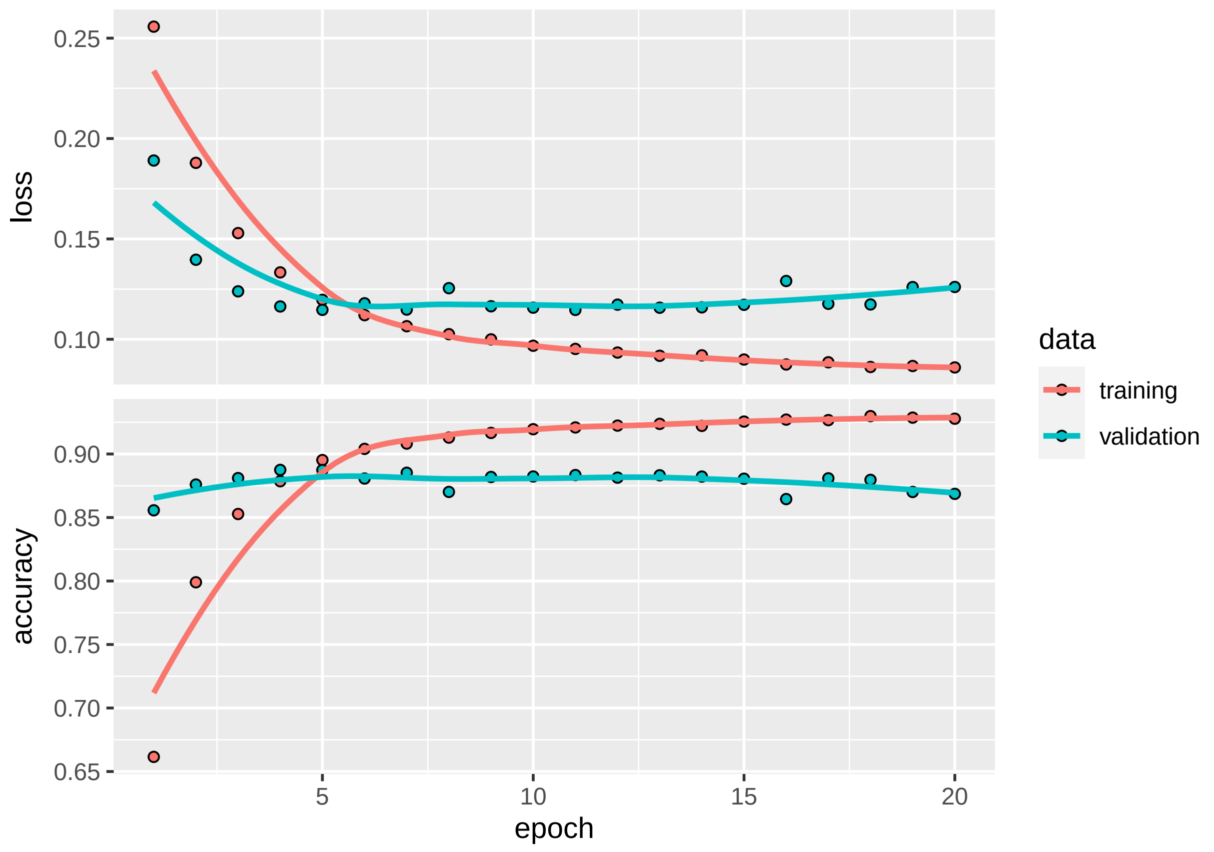
```
##               [,1]
##  [1,] 0.004569453
##  [2,] 0.999456465
##  [3,] 0.541818380
##  [4,] 0.960826993
##  [5,] 0.998723686
##  [6,] 0.990002096
##  [7,] 0.998870730
##  [8,] 0.001100726
##  [9,] 0.984973669
## [10,] 0.999403238
```

Tanh also returns units that are more extreme in confidence and much like how we have seen issues with an increase in loss and a decrease in accuracy with the previous alterations.

Question 5- Preform better on Validation

```
library(keras)
l2_regular_model <- keras_model_sequential() %>% layer_dense(units = 16, kernel_regularizer = regularize
l2_regular_model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
```

```
l2_model_hist <- l2_regular_model %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_test, y_test)
)
plot(l2_model_hist)
```



Results - 4 epochs are the optimal number of epochs

With 4 epochs, this model decreases slightly in accuracy but reduces loss greatly.