Assignment 1

Hannah Cronin

2023-02-19

The following code comes from the Deep Learning with R Notebook. I'm including an untouch control version for comparison purposes. I am using MODELA (4 epochs to compare against).

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

```
str(train_data[[1]])
```

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

```
train_labels[[1]]
```

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

```
vectorize_sequences <- function(sequences, dimension = 10000) {
  # Create an all-zero matrix of shape (len(sequences), dimension)
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for (i in 1:length(sequences))
    # Sets specific indices of results[i] to 1s
    results[i, sequences[[i]]] <- 1
  results
}
# vectorized training data
x_train <- vectorize_sequences(train_data)
# vectorized test data
x_test <- vectorize_sequences(test_data)
```

```
str(x_train[1,])
```

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

```
# vectorized labels
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)
```

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

```
model %>% compile(
  optimizer = optimizer_rmsprop(learning_rate=0.001),
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

_

`{r} --Commenting this code chunk out and passing each as strings instead of objects. model %>% compile(   optimizer = optimizer_rmspr`

```
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]
```

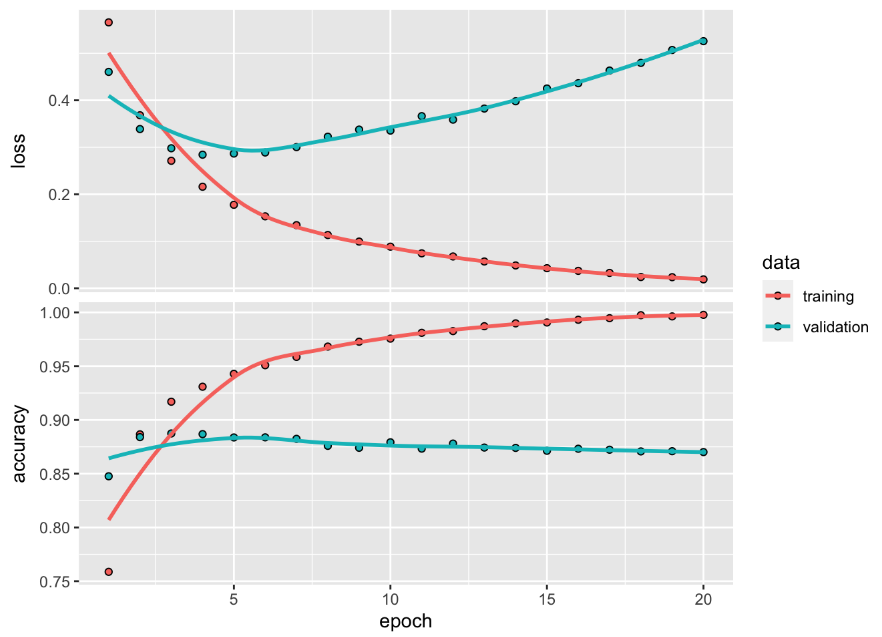
```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
```

```
str(history)
```

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.566 0.368 0.271 0.216 0.178 ...
##   ..$ accuracy    : num [1:20] 0.759 0.887 0.917 0.931 0.943 ...
##   ..$ val_loss    : num [1:20] 0.46 0.339 0.298 0.284 0.287 ...
##   ..$ val_accuracy: num [1:20] 0.848 0.884 0.887 0.887 0.884 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
result <- model %>% evaluate(x_test, y_test)
```

```
plot(history)
```



```
modela <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
modela %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
modela %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- modela %>% evaluate(x_test, y_test)
```

```
results
```

```
##      loss  accuracy
## 0.2836832 0.8879600
```

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

```
##              [,1]
##  [1,] 0.195285246
##  [2,] 0.999293804
##  [3,] 0.876297951
##  [4,] 0.862509727
##  [5,] 0.959679663
##  [6,] 0.800767243
##  [7,] 0.999716699
##  [8,] 0.008087011
##  [9,] 0.963101804
## [10,] 0.992136538
```

End of “control” code chunk

1. Using 3 Hidden Layers

```
model1 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 16, activation = "relu") %>% #New hidden layer
  layer_dense(units = 1, activation = "sigmoid")

model1 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
history1 <- model1 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 4,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
results1 <- model1 %>% evaluate(x_test, y_test)
```

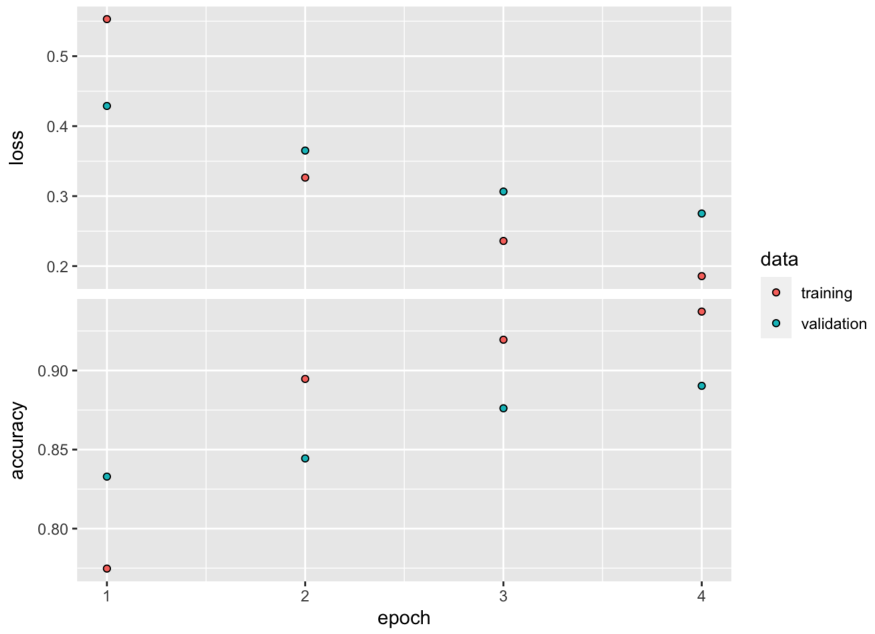
```
str(history1)
```

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 4
##   ..$ steps  : int 30
##  $ metrics:List of 4
##   ..$ loss        : num [1:4] 0.553 0.326 0.236 0.186
##   ..$ accuracy    : num [1:4] 0.775 0.895 0.919 0.937
##   ..$ val_loss    : num [1:4] 0.429 0.365 0.307 0.275
##   ..$ val_accuracy: num [1:4] 0.833 0.844 0.876 0.89
##  - attr(*, "class")= chr "keras_training_history"
```

```
results1
```

```
##      loss  accuracy
## 0.2911662 0.8822800
```

```
plot(history1)
```



```
# After adding another hidden layer, we can see that loss rises rises. With 3 hidden layers, the model is worse a
t predicting the data and it's targets.
# Test Accuracy decreases.
# I would recommend dropped the number of epochs down to 3 to avoid overfitting.
```

2. Increase the # of hidden units

3. Using MSE loss function

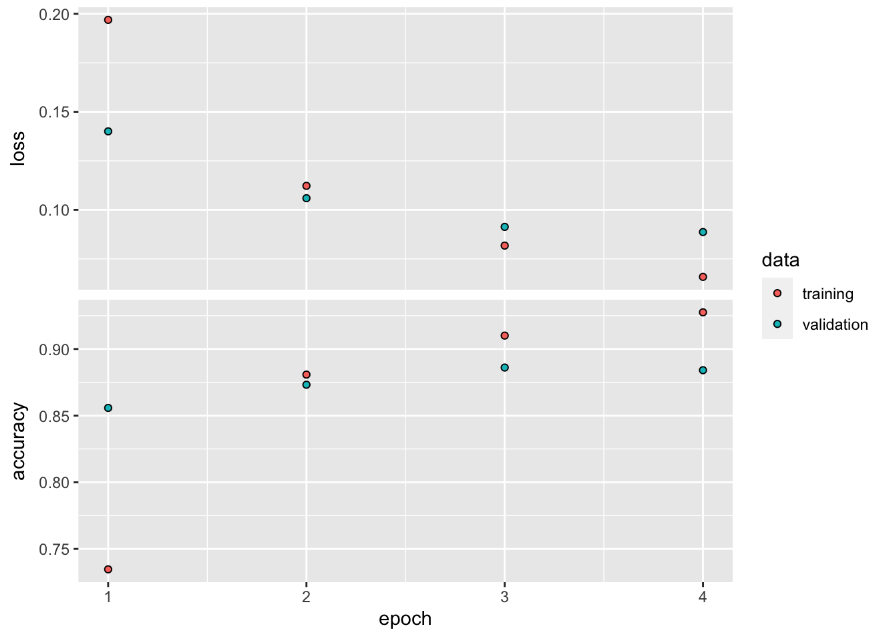
```r model3 <- keras_model_sequential() %>% layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>% layer_dense(units = 16, activation = "relu'

model3 %>% compile( optimizer = "rmsprop", loss = "mse", #Loss function of MSE metrics = c("accuracy") ) history3 <- model3 %>% fit( partial_x_train, partial_

```
r results3
```

```
##      loss  accuracy ## 0.0917963 0.8801600
```

```
r plot(history3)
```


loss
accuracy
epoch
data
training
validation

```
r # With the MSE function, loss is lower as compared to Binary Crossentropy, but Binary Crossentropy is the better option due to the
```

4. Using TANH activation
```

```
model4 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "tanh")
  layer_dense(units = 1, activation = "sigmoid")
```

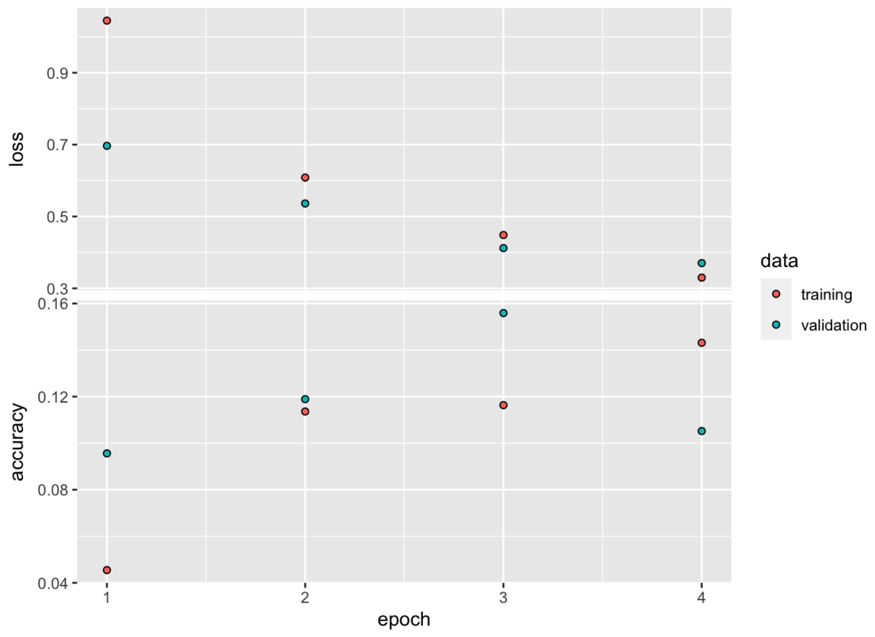
```
## <keras.layers.core.dense.Dense object at 0x133180490>
```

```
model4 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
history4 <- model4 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 4,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
results4 <- model4 %>% evaluate(x_test, y_test)
```

```
results4
```

```
##      loss  accuracy
## 0.3873087 0.0976400
```

```
plot(history4)
```



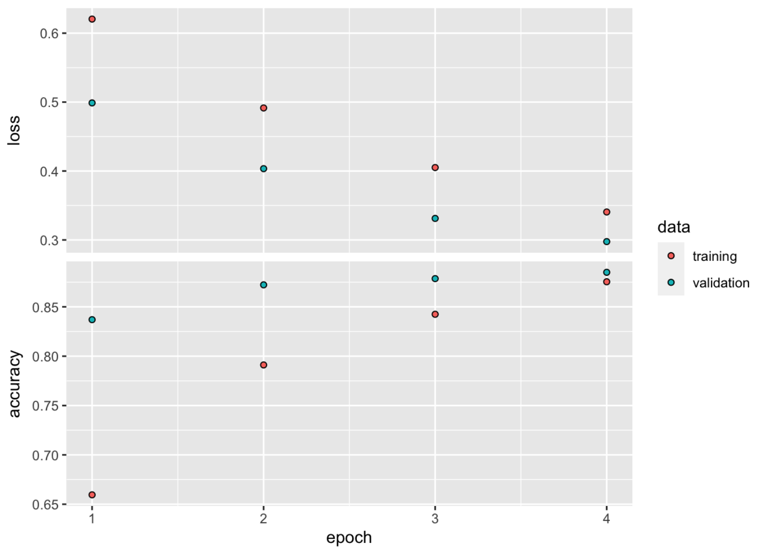
```
# This model is a much poorer choice due to the drastically increased loss and decreased accuracy.
# Relu as an activation function is much better and avoids the problems caused by TANH.
```

5. Using dropout

```r model5 <- keras_model_sequential() %>% layer_dense(units = 16, activation = “relu”, input_shape = c(10000)) %>% layer_dropout(0.4) %>% layer_dense(units = 16, activation = “relu”) %>% layer_dropout(0.4) %>% layer_dense(units = 1, activation = “sigmoid”)

model5 %>% compile( optimizer = “rmsprop”, loss = “binary_crossentropy”, metrics = c(“accuracy”) ) history5 <- model5 %>% fit( partial_x_train, partial_y_train, epochs = 4, batch_size = 512, validation_data = list(x_val, y_val) ) results5 <- model5 %>% evaluate(x_test, y_test) # I tried using both 0.3 and 0.5 and the dropout of 0.4 yielded the best results. ```

```
r plot(history5)
```
```



```
r results5
```

```
##      loss  accuracy ## 0.3071659 0.8819200
```

A model with multiple changes

```
model6 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dropout(0.4) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dropout(0.4) %>%
  layer_dense(units = 1, activation = "sigmoid")

model6 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history6 <- model6 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 4,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
results6 <- model6 %>% evaluate(x_test, y_test)
```

While changes multiple things on the original model, the was the best combinations of things I was able to find. I've run through many versions of this model with variations of the numbers, but have not been able to get the model to increase in accuracy. The MSE makes the loss lower, but I'm not sure I agree with it's usage in this model (I mentioned the reason above). I was interested to see how changing multiple things on the same model would react, but I have not been able to improve upon the original.

Note: I avoided using actual figures for my comparison descriptions the numbers change when I go to knit the R Markdown file. Due to the slight randomizations, the numbers are similar but never the same with each run.