Homework 9

Due Wednesday Nov 20, 2019

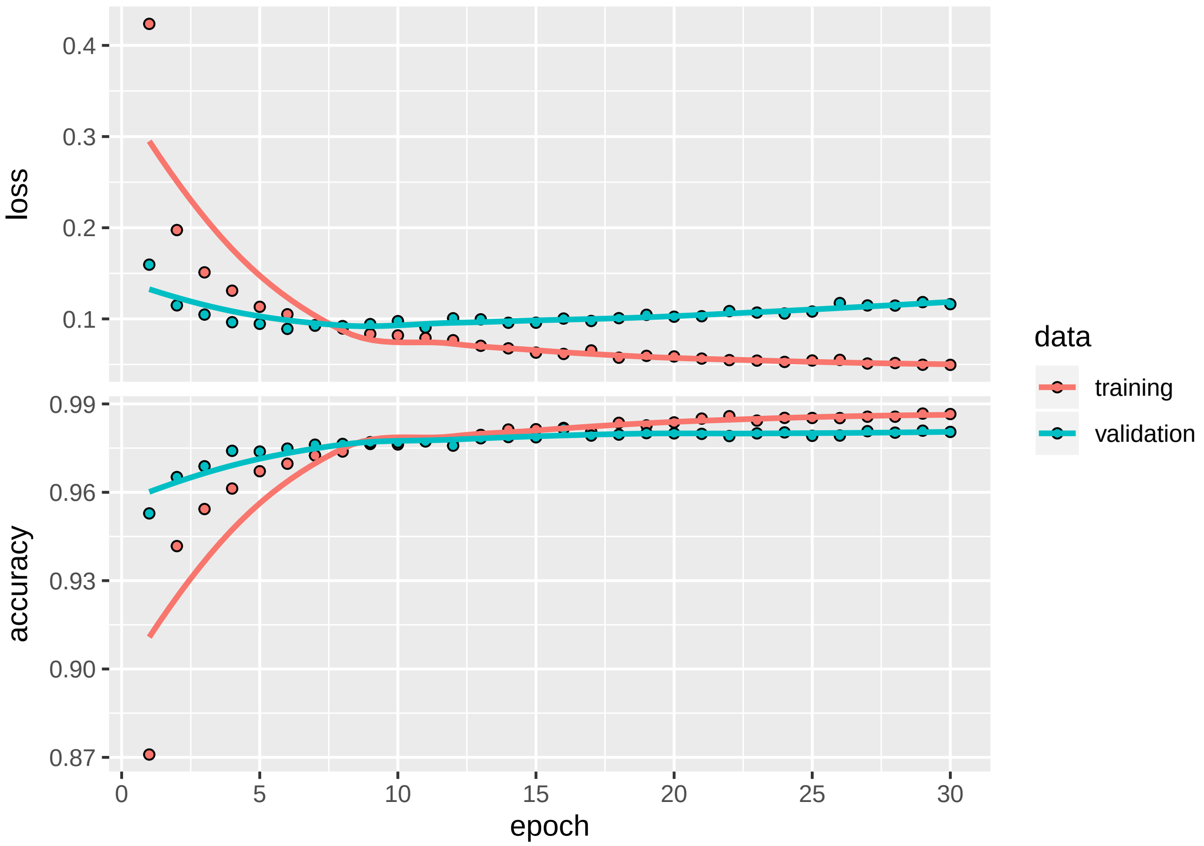
2019-11-17

```r
#devtools::install_github("rstudio/keras")
#install_keras()

#install_tensorflow(gpu = FALSE)

mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# reshape
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
# rescale
x_train <- x_train / 255
x_test <- x_test / 255

y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')

summary(model)
```

```
## Model: "sequential"
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense (Dense)                    (None, 256)                   200960
## ___________________________________________________________________________
## dropout (Dropout)                (None, 256)                   0
## ___________________________________________________________________________
## dense_1 (Dense)                  (None, 128)                   32896
## ___________________________________________________________________________
## dropout_1 (Dropout)              (None, 128)                   0
## ___________________________________________________________________________
## dense_2 (Dense)                  (None, 10)                    1290
## ===========================================================================
## Total params: 235,146
```

```
## Trainable params: 235,146
## Non-trainable params: 0
## ___________________________________________________________________________
```

```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

history <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)

plot(history)
```




```
table(model %>% evaluate(x_test, y_test))
```

```
##                    accuracy
## loss                0.981000006198883
##   0.107427766953344                 1
```

```
table(model %>% predict_classes(x_test))
```

```
##
##    0    1    2    3    4    5    6    7    8    9
```

```
##  986 1145 1038 1009  991  893  958 1025  962  993
```

```
# https://keras.rstudio.com/articles/examples/mnist_cnn.html
```

Example 1

Basic Classification

```
fashion_mnist <- dataset_fashion_mnist()

c(train_images, train_labels) %<-% fashion_mnist$train
c(test_images, test_labels) %<-% fashion_mnist$test

class_names = c('T-shirt/top',
                'Trouser',
                'Pullover',
                'Dress',
                'Coat',
                'Sandal',
                'Shirt',
                'Sneaker',
                'Bag',
                'Ankle boot')

dim(train_images)
```

```
## [1] 60000    28    28
```

```
dim(train_labels)
```

```
## [1] 60000
```

```
train_labels[1:20]
```

```
##  [1] 9 0 0 3 0 2 7 2 5 5 0 9 5 5 7 9 1 0 6 4
```

```
dim(test_images)
```

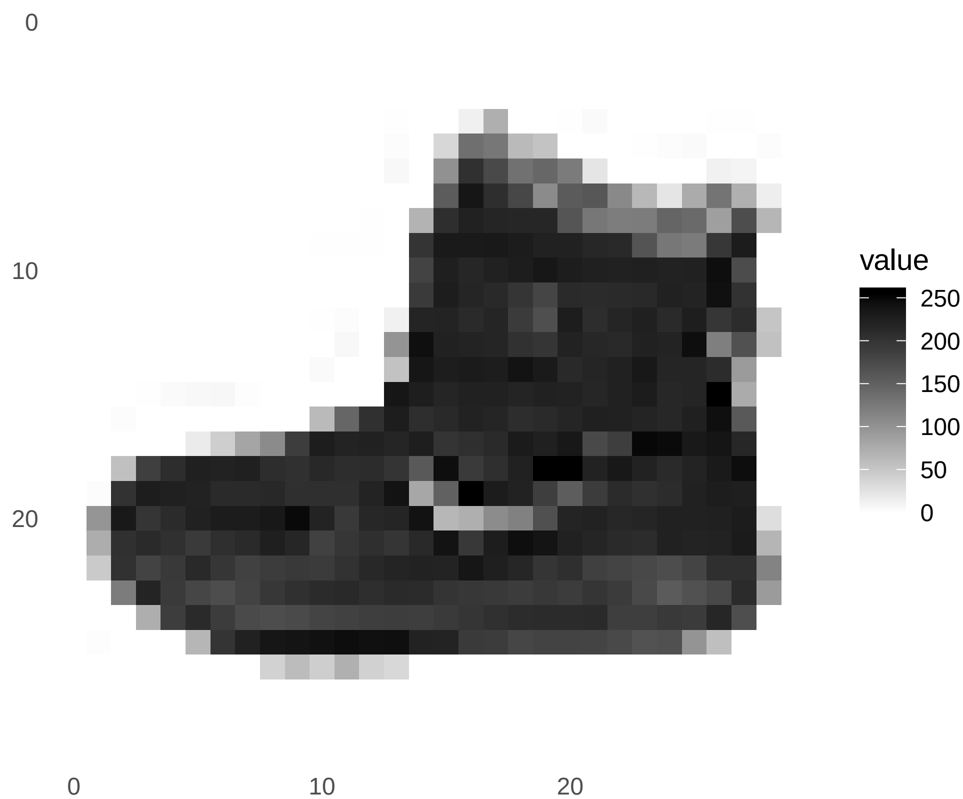
```
## [1] 10000    28    28
```

```
dim(test_labels)
```

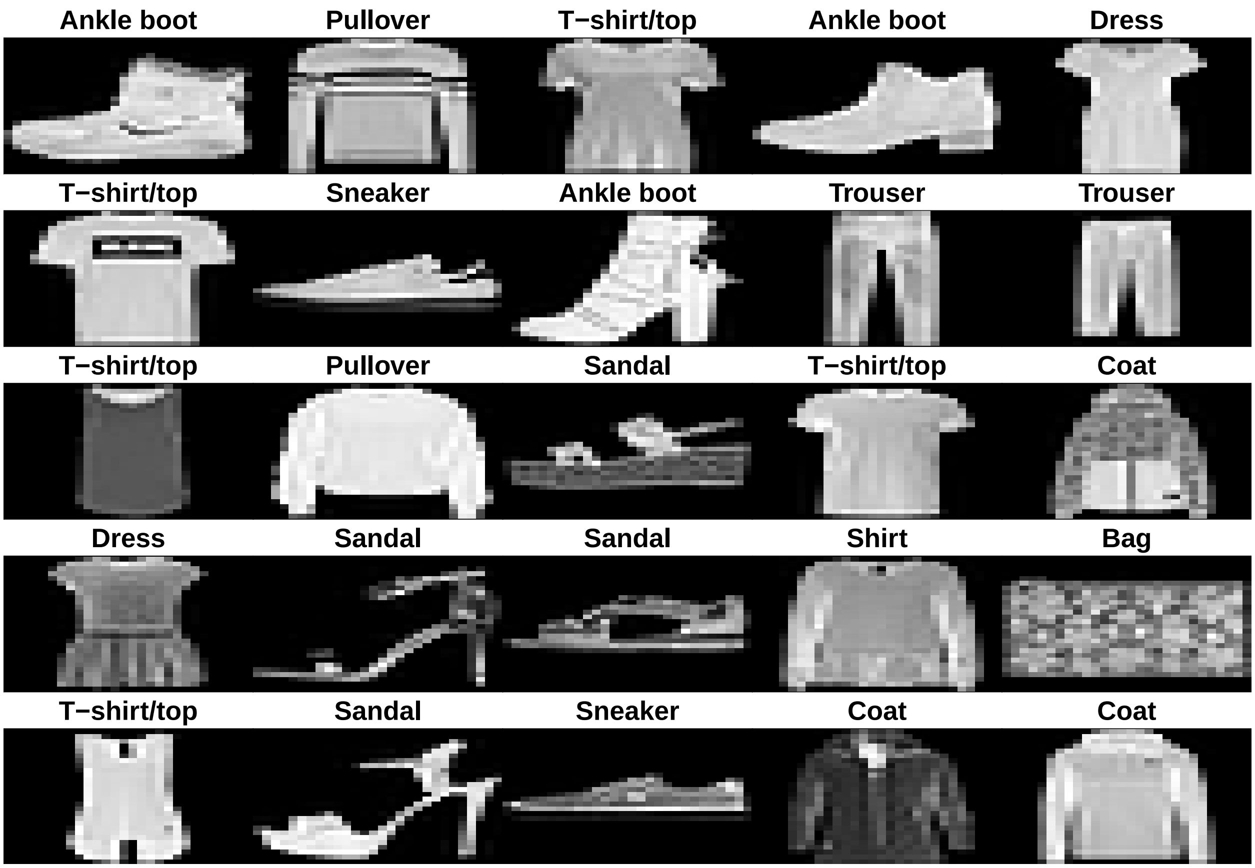
```
## [1] 10000
```

```
image_1 <- as.data.frame(train_images[1, , ])
colnames(image_1) <- seq_len(ncol(image_1))
image_1$y <- seq_len(nrow(image_1))
image_1 <- gather(image_1, "x", "value", -y)
image_1$x <- as.integer(image_1$x)

ggplot(image_1, aes(x = x, y = y, fill = value)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "black", na.value = NA) +
  scale_y_reverse() +
  theme_minimal() +
  theme(panel.grid = element_blank())   +
```

```
  theme(aspect.ratio = 1) +
  xlab("") +
  ylab("")
```




```
train_images <- train_images / 255
test_images <- test_images / 255

par(mfcol=c(5,5))
par(mar=c(0, 0, 1.5, 0), xaxs='i', yaxs='i')
for (i in 1:25) {
  img <- train_images[i, , ]
  img <- t(apply(img, 2, rev))
  image(1:28, 1:28, img, col = gray((0:255)/255), xaxt = 'n', yaxt = 'n',
        main = paste(class_names[train_labels[i] + 1]))
}
```




```r
model <- keras_model_sequential()
model %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

model %>% compile(
  optimizer = 'adam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)

model %>% fit(train_images, train_labels, epochs = 5)

score <- model %>% evaluate(test_images, test_labels)

cat('Test loss:', score$loss, "\n")
```

```
## Test loss: 0.3642727
```

```r
cat('Test accuracy:', score$acc, "\n")
```

```
## Test accuracy: 0.8708
```

```r
predictions <- model %>% predict(test_images)

predictions[1, ]
```

```
##  [1] 1.091267e-06 1.706949e-07 6.287783e-08 9.116476e-09 2.204292e-07
##  [6] 2.915844e-03 3.226610e-07 8.017538e-02 5.035471e-05 9.168565e-01
```

```r
which.max(predictions[1, ])
```

```
## [1] 10
```

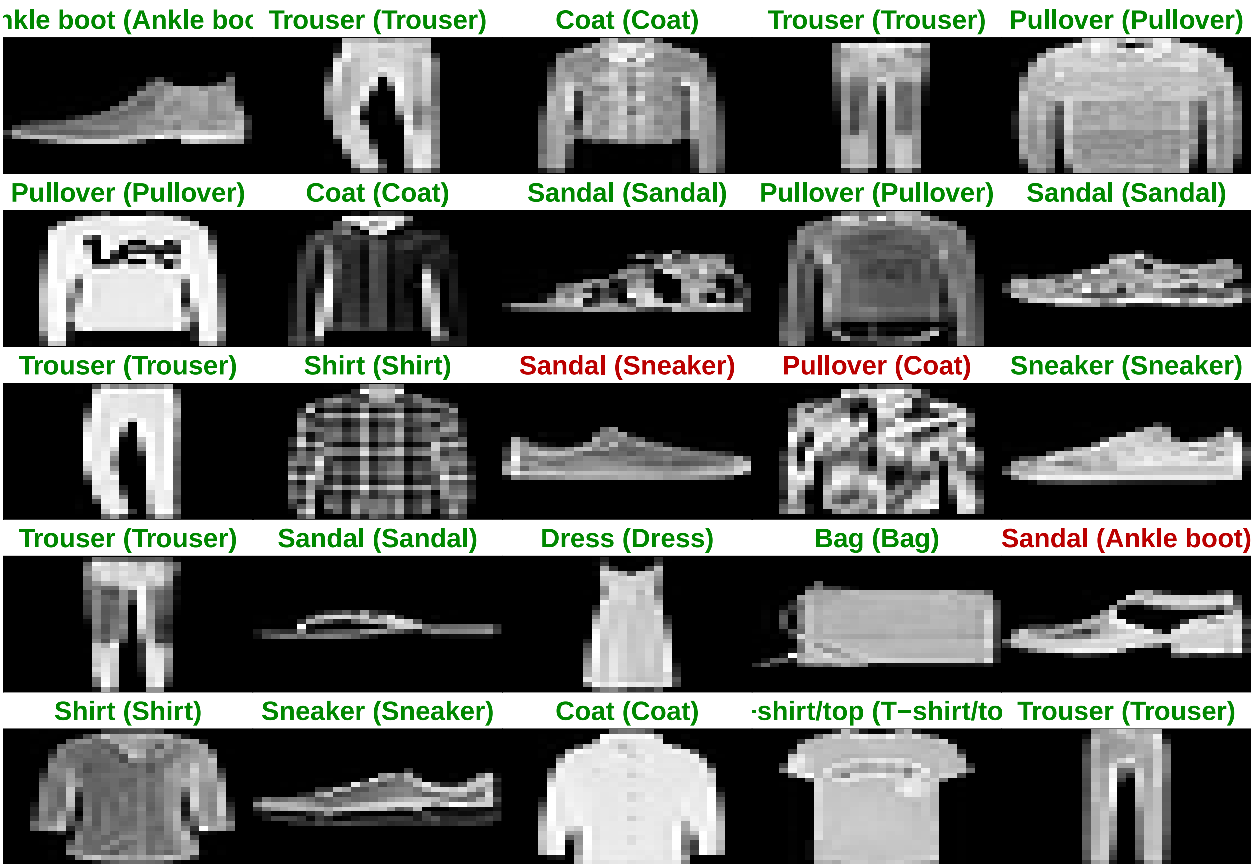
```r
class_pred <- model %>% predict_classes(test_images)
class_pred[1:20]
```

```
##  [1] 9 2 1 1 6 1 4 6 5 7 4 5 5 3 4 1 2 2 8 0
```

```r
test_labels[1]
```

```
## [1] 9
```

```r
par(mfcol=c(5,5))
par(mar=c(0, 0, 1.5, 0), xaxs='i', yaxs='i')
for (i in 1:25) {
  img <- test_images[i, , ]
  img <- t(apply(img, 2, rev))
  # subtract 1 as labels go from 0 to 9
  predicted_label <- which.max(predictions[i, ]) - 1
  true_label <- test_labels[i]
  if (predicted_label == true_label) {
    color <- '#008800'
  } else {
    color <- '#bb0000'
  }
  image(1:28, 1:28, img, col = gray((0:255)/255), xaxt = 'n', yaxt = 'n',
        main = paste0(class_names[predicted_label + 1], " (",
                      class_names[true_label + 1], ")"),
        col.main = color)
}
```




```
# Grab an image from the test dataset
# take care to keep the batch dimension, as this is expected by the model
img <- test_images[1, , , drop = FALSE]
dim(img)
```

```
## [1]  1 28 28
```

```
predictions <- model %>% predict(img)
predictions
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 1.091267e-06 1.706949e-07 6.287807e-08 9.116476e-09 2.204288e-07
##             [,6]        [,7]       [,8]         [,9]     [,10]
## [1,] 0.002915846 3.22661e-07 0.08017541 5.035471e-05 0.9168565
```

```
# subtract 1 as labels are 0-based
prediction <- predictions[1, ] - 1
which.max(prediction)
```

```
## [1] 10
```

```
class_pred <- model %>% predict_classes(img)
class_pred
```

```
## [1] 9
```

Example 2

Text Classification

```
imdb <- dataset_imdb(num_words = 10000)

c(train_data, train_labels) %<-% imdb$train
c(test_data, test_labels) %<-% imdb$test

word_index <- dataset_imdb_word_index()

paste0("Training entries: ", length(train_data), ", labels: ", length(train_labels))
```

```
## [1] "Training entries: 25000, labels: 25000"
```

```
length(train_data[[1]])
```

```
## [1] 218
```

```
length(train_data[[2]])
```

```
## [1] 189
```

```
word_index_df <- data.frame(
  word = names(word_index),
  idx = unlist(word_index, use.names = FALSE),
  stringsAsFactors = FALSE
)

# The first indices are reserved
word_index_df <- word_index_df %>% mutate(idx = idx + 3)
word_index_df <- word_index_df %>%
  add_row(word = "<PAD>", idx = 0)%>%
  add_row(word = "<START>", idx = 1)%>%
  add_row(word = "<UNK>", idx = 2)%>%
  add_row(word = "<UNUSED>", idx = 3)

word_index_df <- word_index_df %>% arrange(idx)

decode_review <- function(text){
  paste(map(text, function(number) word_index_df %>%
              filter(idx == number) %>%
              select(word) %>%
              pull()),
        collapse = " ")
}

decode_review(train_data[[1]])
```

```
## [1] "<START> this film was just brilliant casting location scenery story direction everyone's really
```

```
train_data <- pad_sequences(
  train_data,
  value = word_index_df %>% filter(word == "<PAD>") %>% select(idx) %>% pull(),
  padding = "post",
  maxlen = 256
)
```

```r
test_data <- pad_sequences(
  test_data,
  value = word_index_df %>% filter(word == "<PAD>") %>% select(idx) %>% pull(),
  padding = "post",
  maxlen = 256
)

length(train_data[1, ])
```

```
## [1] 256
```

```r
length(train_data[2, ])
```

```
## [1] 256
```

```r
# input shape is the vocabulary count used for the movie reviews (10,000 words)
vocab_size <- 10000

model <- keras_model_sequential()
model %>%
  layer_embedding(input_dim = vocab_size, output_dim = 16) %>%
  layer_global_average_pooling_1d() %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% summary()
```

```
## Model: "sequential"
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## embedding (Embedding)            (None, None, 16)              160000
## ___________________________________________________________________________
## global_average_pooling1d (Global (None, 16)                    0
## ___________________________________________________________________________
## dense (Dense)                    (None, 16)                    272
## ___________________________________________________________________________
## dense_1 (Dense)                  (None, 1)                     17
## ===========================================================================
## Total params: 160,289
## Trainable params: 160,289
## Non-trainable params: 0
## ___________________________________________________________________________
```

```r
model %>% compile(
  optimizer = 'adam',
  loss = 'binary_crossentropy',
  metrics = list('accuracy')
)

x_val <- train_data[1:10000, ]
partial_x_train <- train_data[10001:nrow(train_data), ]

y_val <- train_labels[1:10000]
partial_y_train <- train_labels[10001:length(train_labels)]
```

```r
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 40,
  batch_size = 512,
  validation_data = list(x_val, y_val),
  verbose=1
)

results <- model %>% evaluate(test_data, test_labels)
results
```

```
## $loss
## [1] 0.3400047
##
## $accuracy
## [1] 0.8702
```

```r
plot(history)
```