Homework 9

Adeline Guthrie

11/16/2019

Example 1: MNIST example with handwritten digits

```
# Preparing the data

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)
```

```
# Defining the model

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)
```

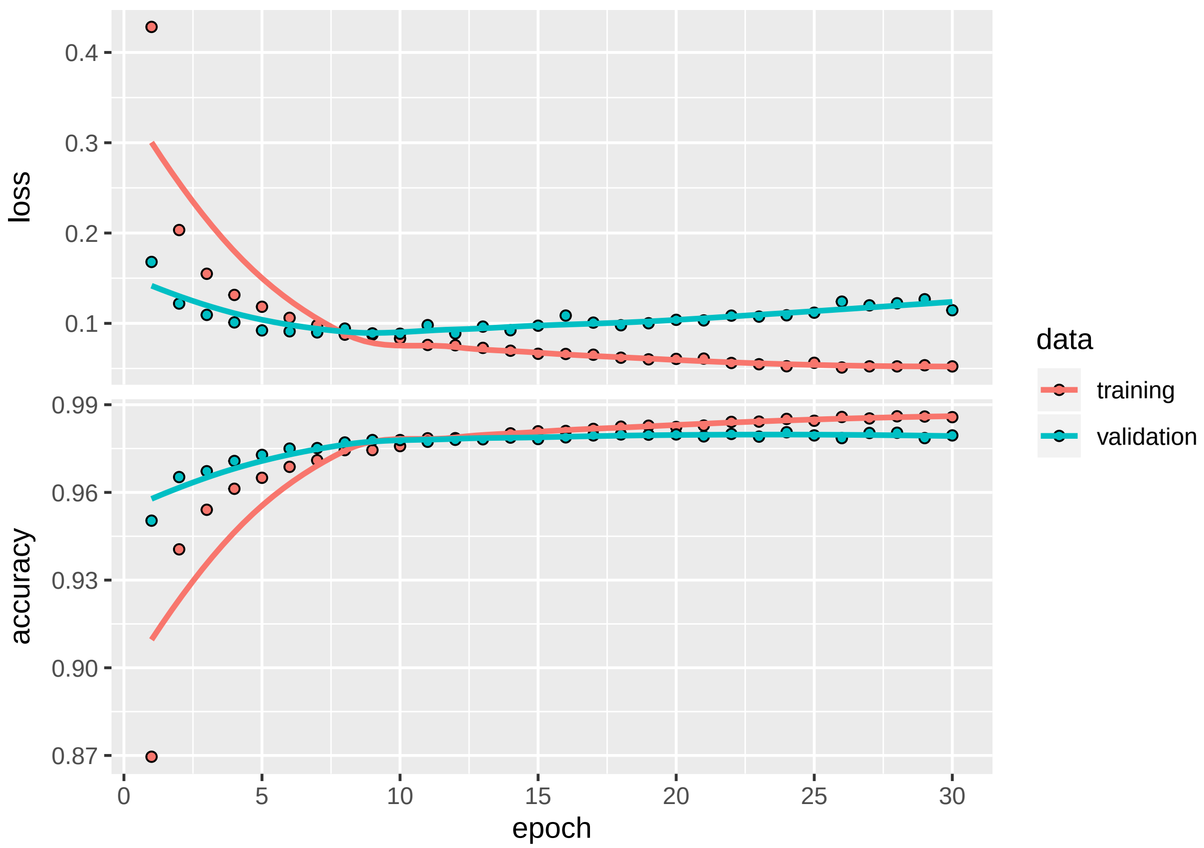
```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense (Dense)                       (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')
)
```

```
# Training and evaluation
history <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
```

```
plot(history)
```



Example 2: Tutorial with MNIST fashion dataset

```
# Import data
fashion_mnist <- dataset_fashion_mnist()

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

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

```r
# Explore the Data
dim(train_images)
```

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

```r
dim(train_labels)
```

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

```r
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
```

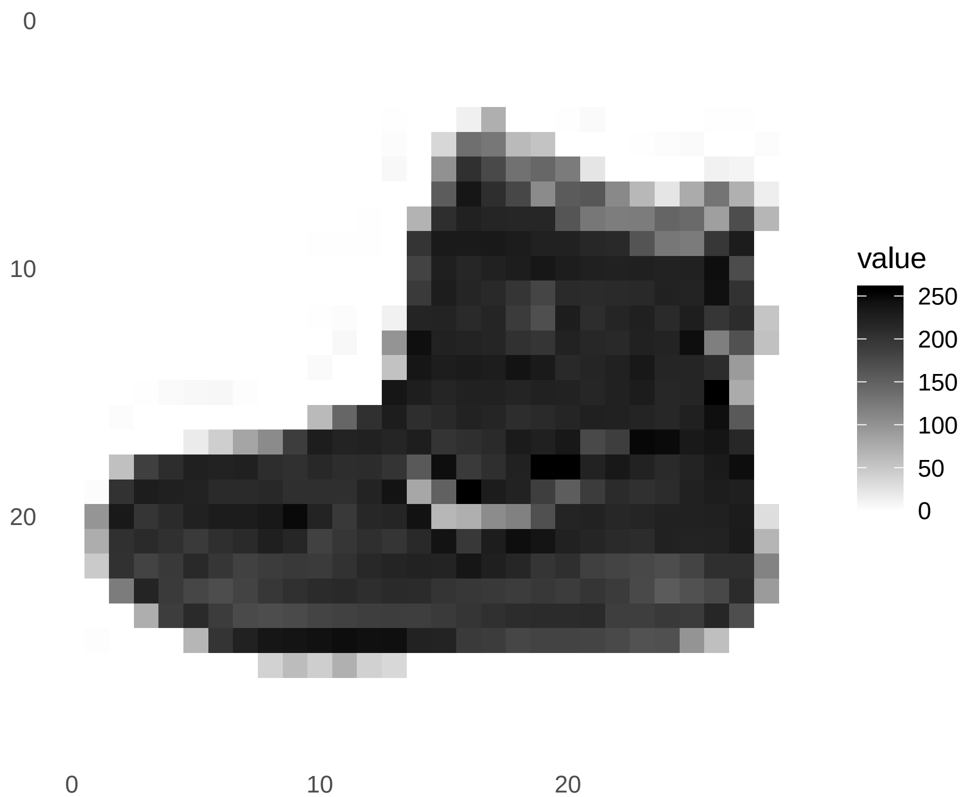
```r
dim(test_images)
```

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

```r
dim(test_labels)
```

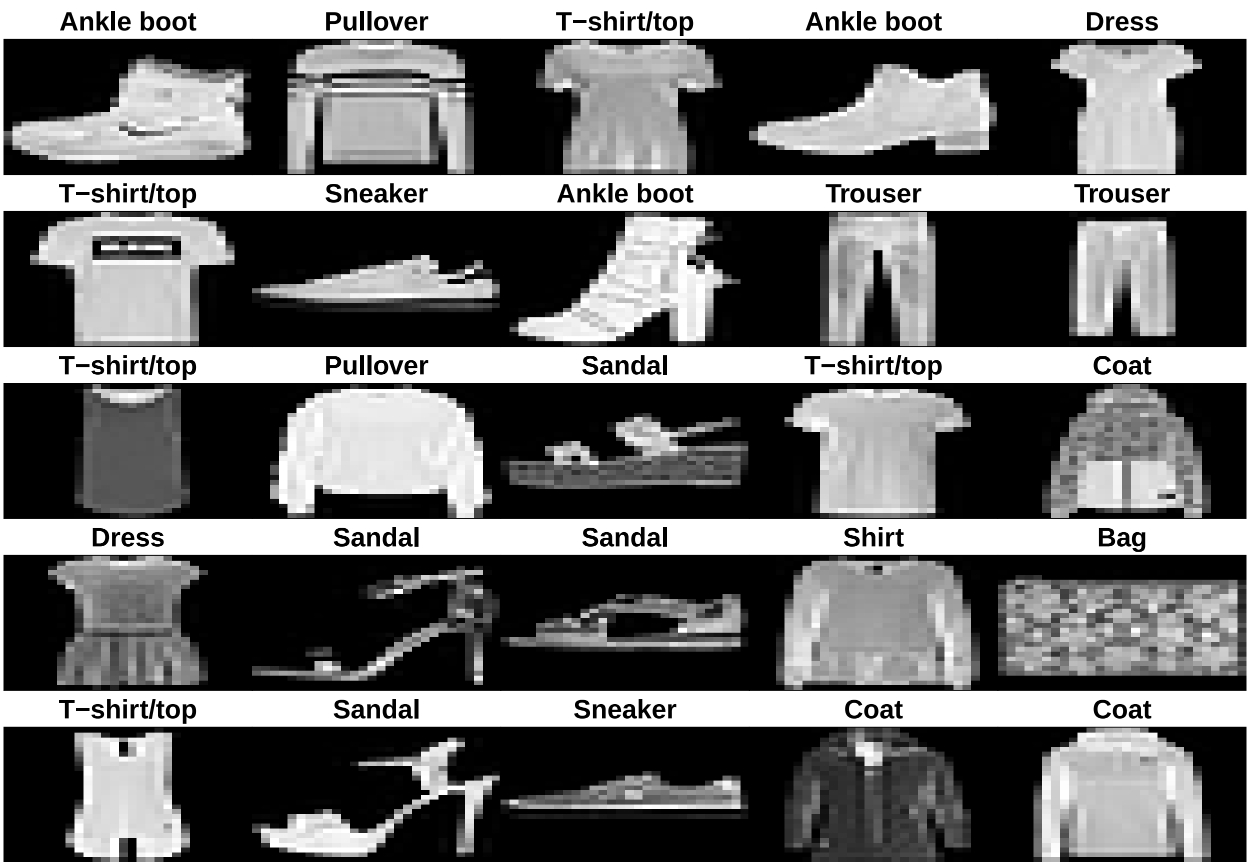
```
## [1] 10000
```

```r
# Preprocess the data

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
# Build the model

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

```r
# Compile the model
model %>% compile(
  optimizer = 'adam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)
```

```r
# Train the model
model %>% fit(train_images, train_labels, epochs = 5)
```

```r
# Evaluate accuracy
score <- model %>% evaluate(test_images, test_labels)

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

```
## Test loss: 0.3453463
```

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

```
## Test accuracy: 0.8756
```

```r
# Make predictions
predictions <- model %>% predict(test_images)
predictions[1, ]
```

```
##  [1] 6.618588e-07 8.337598e-09 1.097081e-06 2.366953e-08 1.600139e-06
##  [6] 9.270784e-03 1.830074e-06 2.304642e-02 3.765865e-05 9.676399e-01
```

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

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

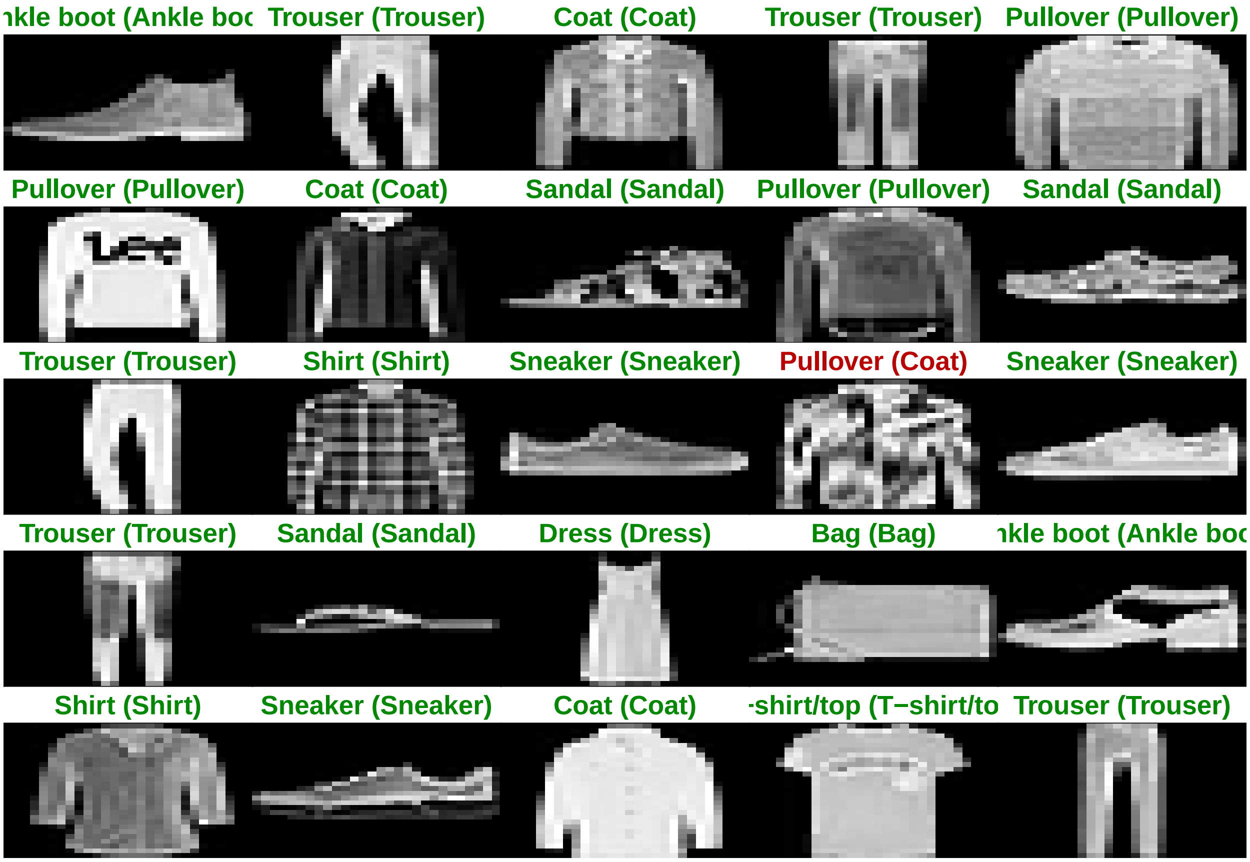
```r
class_pred <- model %>% predict_classes(test_images)
class_pred[1:20]
```

```
##  [1] 9 2 1 1 6 1 4 6 5 7 4 5 7 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)
}
```




```r
# 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
```

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

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 6.618588e-07 8.337567e-09 1.097081e-06 2.366948e-08 1.600139e-06
##             [,6]         [,7]      [,8]         [,9]     [,10]
## [1,] 0.009270779 1.830067e-06 0.0230464 3.765854e-05 0.9676399
```

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

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

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

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