# DATA 300 3 Homework 6 Solution

Aadarsha Gopala Reddy

December 7, 2022

## Contents



## 1. Setup

Load the packages "keras" and "tensorflow." If this is your first time using these packages, you may have to install the underlying python libraries using the following functions (these functions do not take any arguments):

- install_tensorflow()
- install_keras()

Note: Windows machines may experience an OpenSSL error when attempting to install these packages. This is caused by a bug in the reticulate package, which is automatically loaded when using python libraries. You can work around this problem by installing a developer version of the reticulate package, which has implemented a bug fix:

```
remotes::install_github("rstudio/reticulate")
```

`library()` imports the packages into the R session.

```
knitr::opts_chunk$set(echo = TRUE)
library(reticulate)
library(tensorflow)
library(keras)
#install_tensorflow()
#install_keras()
```

```
#it kinda works when I run the install commands in the console, but when
#knitting, I need to comment them out, else I run out of memory
```

---

## 2. Load the Dataset

Load the Fashion-MNEST data set using the function `dataset_fashion_mnest()`. Store this data set as an object in R. (Unlike data sets we have seen in the past, this data set is a list containing multiple sub-objects.)

`dataset_fashion_mnist()` returns a list containing the following sub-objects:

- `x_train`: training data
- `y_train`: training labels
- `x_test`: test data
- `y_test`: test labels

```
# load the data set
fashion_mnist <- dataset_fashion_mnist()
```

```
## Loaded Tensorflow version 2.9.3
```

---

## 3. Format the Dataset

Using the data set that you loaded in the previous question, create four objects in R:

- Image data for the training data set;
- Class labels for the training data set;
- Image data for the testing data set;
- Class labels for the testing data set.

Hint: use the dollar sign operator to select individual components of the Fashion-MNEST data set.

```
# format the data set
train_images <- fashion_mnist$train$x
train_labels <- fashion_mnist$train$y
test_images <- fashion_mnist$test$x
test_labels <- fashion_mnist$test$y
```

---

## 4. Recode Class Labels

The class labels for both the testing and training data set are stored as numeric values 0 through 9. Recode the class labels into two categories: one for items worn on the torso, and another for pants/shoes/accessories. In your new coding, T-shirts/tops, pullovers, dresses, coats, and shirts will be coded 1, and all other classes will be coded 0.

When you are finished, you should have two objects:

- Training class labels, containing 60,000 instances of either 0 or 1;
- Testing class labels, containing 10,000 instances of either 0 or 1.

```
# recode the class labels
train_labels <- ifelse(train_labels %in% c(0, 1, 2, 3, 4, 5), 1, 0)
test_labels <- ifelse(test_labels %in% c(0, 1, 2, 3, 4, 5), 1, 0)
```

---

## 5. Pre-Process Pixel Values

The pixel brightness values are stored as numeric values between 0 and 255. Re-scale these values to be bounded between 0 and 1. (This will make the computations easier for our neural net.) Values of 255 in the original coding will become values of 1 in the new coding; values of 100 in the old coding will become values of .392 in the new coding; values of 0 in the original coding will become values of 0 in the new coding; and so on.

Hint: you can accomplish this by dividing every value by 255.

```
# pre-process the pixel values
train_images <- train_images / 255
test_images <- test_images / 255
```

---

## 6. Structure a Neural Network

Using the keras package, you can build your network structure layer-by-layer sequentially. Begin by initializing a sequential neural network:

```
net <- keras_model_sequential()
```

Add layers to your model using a pipe. The code below adds the first layer (a convolutional layer) to your network:

```
net %>% layer_conv_2d(input_shape = c(28, 28, 1), filters = 1, kernel_size = c(15, 15))
```

Add additional layers in order by adding more pipes:

`net %>% [layer1] %>% [layer2] %>%` ... and so on.

Check the layers you have added to your network by typing its name. R will output a line for each layer, the output shape of each layer, and some summary information about the model.

Create a network structure containing the following layers:

- A convolutional layer
  - Arguments:
    - This layer should take an input of a 2-dimensional array of 28x28 pixels;
    - This layer should apply a single convolutional filter to the input;
    - The convolutional filter should be 15x15 pixels;
  - Outputs:
    - This layer should output a 14x14 output feature map.
- A max-pooling layer
  - Arguments:
    - This layer should "pool" values by taking the maximum value within each 2x2 window.
  - Outputs:
    - This layer should output a 7x7 pooled feature map.
- A flattening layer
  - Arguments:
    - This layer should take, as an input, the 7x7 pooled feature map produced by the previous layer.
  - Output:

        * This layer should output a vector of 49 values; these values should be "flattened" in the sense that they are no longer structured as a 2-dimensional array, but simply as a vector of 49 individual numeric values.
- A dense (AKA "fully-connected" or "hidden") layer
  - Arguments:
    * This layer should contain 49 nodes (one for each value it will take as an input);
    * This layer should process these values using the "relu" activation function. (The ReLU, or rectified linear unit, function is a popular function in neural networks; it changes negative values to 0, which speeds up computation time and introduces non-linearity into the network.)
  - Output:
    * This layer should output a vector of 49 numeric values (one for each node).
- An output layer: a dense layer that generates an output value
  - Arguments:
    * This layer should contain one single node;
    * This layer should use a "sigmoid" activation function to transform its inputs into a single value between 0 and 1.
  - Output:
    * This layer should output a single value bounded between 0 and 1; this is the predicted probability that an image depicts a top.

`layer_conv_2d()` creates a convolutional layer.

`layer_max_pooling_2d()` creates a max-pooling layer.

`layer_flatten()` creates a flattening layer.

`layer_dense()` creates a dense layer.

```
# structure the neural network
net <- keras_model_sequential() %>%
  layer_conv_2d(input_shape = c(28, 28, 1), filters = 1, kernel_size = c(15, 15)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 49, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

---

## 7. Compile the Network

Compile the network that you created in the previous step.

`net %>% compile(...)`

The compile function should take the following arguments:

- The compiler should use the "adam" optimizer;
- The compiler should use the "binary_crossentropy" loss function;
- The compiler should use the "accuracy" metric.

`compile()` is a function that takes the following arguments:

- `optimizer`: This is the optimizer that the network will use to update its weights.
- `loss`: This is the loss function that the network will use to evaluate its performance.
- `metrics`: This is the metric that the network will use to evaluate its performance.

```
# compile the network
net %>% compile(optimizer = "adam", loss = "binary_crossentropy", metrics = "accuracy")
```

---

## 8. Train the Network

Train your neural network on the images and class labels for the training data set.

`net %>% fit(...)`

`fit()` is a function that takes the following arguments:

- `x`: This is the input data that the network will use to train itself.
- `y`: This is the output data that the network will use to train itself.
- `epochs`: This is the number of times that the network will train itself on the entire training data set.

```
# train the network
net %>% fit(train_images, train_labels, epochs = 7)
```

---

## 9. Test the Network

Find the accuracy of your neural network in both the testing and the training data.

`net %>% evaluate(...)`

What percentage of the training images does your neural network classify correctly? What percentage of the testing images does your neural network classify correctly?

`evaluate()` is a function that takes the following arguments:

- `x`: This is the input data that the network will use to test itself.
- `y`: This is the output data that the network will use to test itself.

```
# test the network
net %>% evaluate(test_images, test_labels)
```

```
##      loss  accuracy
## 0.2375883 0.9021000
```

```
net %>% evaluate(train_images, train_labels)
```

```
##      loss  accuracy
## 0.2062081 0.9142500
```

***The neural network can correctly classify 90.67% and 91.44% of the images from testing and training data sets respectively. (percentages may vary after knit-to-pdf)***

---