MSCI 446: Assignment 4 - Question 1

M. Harper, H. Gommaa, K. Morris

14/04/2021

Include Packages

```
library('tidyverse')
library('caret')
library('ggplot2')
library('gridExtra')
library('glmnet')
library('leaps')

library('tree')
library('rpart')
library('e1071')

theme_set(theme_classic())
```

Quetsion 1: Classification Using NNets

```
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.0.4
```

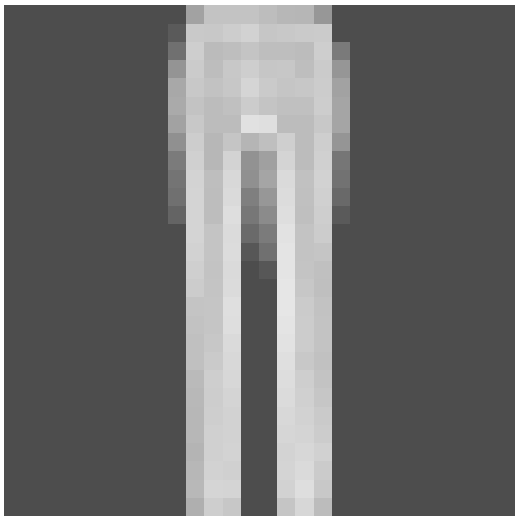
Get the Data

```
mnist <- dataset_fashion_mnist()

x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

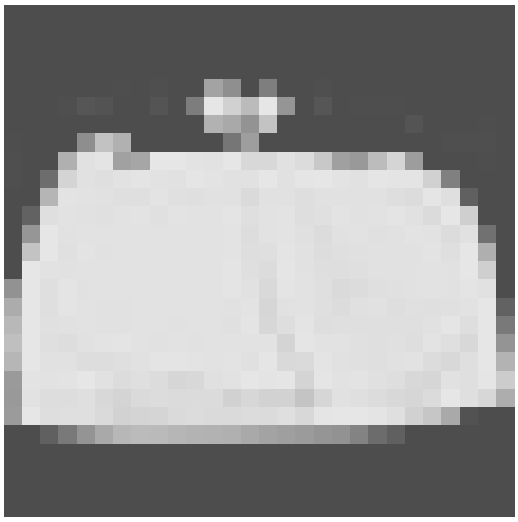
1.2 Plot

```
trouser <- x_train[70, 28:1,1:28]
par(pty='s')
image(t(trouser), col=grey.colors(256), axes=FALSE)
```



Through trail and error, A pair of trousers (class 1) from the test set was found as the 70th instance/data point.

```
bag <- x_train[110, 28:1,1:28]
par(pty='s')
image(t(bag), col=grey.colors(256), axes=FALSE)
```



Through trail and error, A bag (class 9) from the test set was found as the 110th instance/data point.

```
boot <- x_train[1, 28:1,1:28]
par(pty='s')
image(t(boot), col=grey.colors(256), axes=FALSE)
```

Through trail and error, an ankel boot (class 10) from the test set was found as the 1st instance/data point.

1.3 Process the Dataset

```
# Reshape the train and test datasets to consist of 28x28=784 columns (predictor variables)

x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))

# Re-scale the data for the NNet to contain values from 0 to 1 (normalize)

x_train <- x_train /255
x_test <- x_test / 255

# Process the output (y-values) from binary to categorical as there are 10 possible output classes

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

head(y_train, 6)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    0    0    0    0     1
## [2,]    1    0    0    0    0    0    0    0    0     0
```

```
## [3,]    1    0    0    0    0    0    0    0    0    0
## [4,]    0    0    0    1    0    0    0    0    0    0
## [5,]    1    0    0    0    0    0    0    0    0    0
## [6,]    0    0    1    0    0    0    0    0    0    0
```

1.4: Fit a Shallow Network

```
shallow <- keras_model_sequential()

shallow %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dense(units = 10, activation = 'softmax')

#summary(shallow)

shallow %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)

history <- shallow %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2

)

shallow %>% evaluate(x_test, y_test)
```

```
##      loss  accuracy
## 0.3456666 0.8761000
```

Results from Shallow Network:

1. Relu Activation w/ 32 neurons in hidden layer: Test Accuracy = 0.8619
2. Relu Activation w/ 128 neurons in hidden layer: Test Accuracy = 0.8755
3. Relu Activation w/ 256 neurons in hidden layer: Test Accuracy = 0.8824
4. Sigmoid Activation w/ 32 neurons in hidden layer: Test Accuracy = 0.8602
5. Sigmoid Activation w/ 128 neurons in hidden layer: Test Accuracy = 0.8697
6. Sigmoid Activation w/ 256 neurons in hidden layer: Test Accuracy = 0.8713

From the six trails above, it appears as though the model with the best test set performance is the one with *Relu Activation* and *256 Neurons* in the hidden layer.

NOTE: Re-running the shallow network above with the same parameters appears to yield different values for test accuracy each time it is run.

1.5 Fit A Deep Neural Network

```
set.seed(112)

deep <- keras_model_sequential()

deep %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dense(units = 256, activation = 'relu') %>% # deep network
  layer_dense(units = 10, activation = 'softmax')

#summary(shallow)

deep %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)

history <- deep %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2

)

deep %>% evaluate(x_test, y_test)
```

```
##      loss  accuracy
## 0.3480709 0.8842000
```

Deep Network Performance (+1 Additional Hidden Layer): "Hidden Layer = HL"

1. **HL(1)** = 32 neurons, activation = relu, **HL(2)** = 32 neurons, activation = relu. Test Accuracy = 0.8662
2. **HL(1)** = 32 neurons, activation = relu, **HL(2)** = 128 neurons, activation = relu. Test Accuracy = 0.8675
3. **HL(1)** = 32 neurons, activation = relu, **HL(2)** = 256 neurons, activation = relu. Test Accuracy = 0.8717
4. **HL(1)** = 128 neurons, activation = relu, **HL(2)** = 32 neurons, activation = relu. Test Accuracy = 0.8772
5. **HL(1)** = 128 neurons, activation = relu, **HL(2)** = 128 neurons, activation = relu. Test Accuracy = 0.8830
6. **HL(1)** = 128 neurons, activation = relu, **HL(2)** = 256 neurons, activation = relu. Test Accuracy = 0.8737
7. **HL(1)** = 256 neurons, activation = relu, **HL(2)** = 32 neurons, activation = relu. Test Accuracy = 0.8735

8. **HL(1)** = 256 neurons, activation = relu, **HL(2)** = 128 neurons, activation = relu. Test Accuracy = 0.8823

9. **HL(1)** = 256 neurons, activation = relu, **HL(2)** = 256 neurons, activation = relu. Test Accuracy = 0.8835

For the deep neural network above, each of the two hidden layers can either have 32, 128 or 256 neurons, and each layer can either use relu or sigmoid activation functions (for the purposes of this assignment). This results in many possible combinations to test. Our team decided to run 9 tests, changing the combination of the number of hidden layer number of neurons but leaving the activation as relu for both layers.

From the 9 models tested, it appears as though the 9th model, with 256 neurons in both hidden layers, performed the best on the test set. Closely following this models performance was the 5th model tested. This model had 128 neurons in each of the two hidden layers.

NOTE: Re-running the deep network above with the same parameters appears to yield different values for test accuracy each time it is run.