Machine Learning - Exercise 4: Second part

Robin Perälä 175910

5.5.2022

Task 5

Adapted from R-bloggers (2021): Deep Neural Network in R. r-bloggers.com/2021/04/deep-neural. Fit a neural network with a single hidden layer to the Boston Housing data.

```
rm(list=ls()); RNGkind(sample.kind='Rejection')

#Load Library
library(mlbench) #Contains BostonHousing data
library(dplyr) #Contains "mutate_if" function
library(magrittr) #Contains pipes, e.g. %<>%
library(neuralnet) #For fitting neural networks
library(keras) #Contains "keras_model_sequential" function
library(glmnet) #For regularized linear models (e.g. lasso and ridge)

#Getting Data
data("BostonHousing")
data <- BostonHousing
str(data)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : num  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ b      : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
data %<>% mutate_if(is.factor, as.numeric)
data <- as.matrix(data)

#Data Partition
```

```
set.seed(707)
ind <- sample(2, nrow(data), replace = T, prob = c(.7, .3))
training <- data[ind==1, 1:13]
test <- data[ind==2, 1:13]
trainingtarget <- data[ind==1, 14]
testtarget <- data[ind==2, 14]

#Scaling
m <- colMeans(training)
s <- apply(training, 2, sd)
training <- scale(training, center = m, scale = s)
test <- scale(test, center = m, scale = s)

#Model Creation
model <- keras_model_sequential()
model %>%
         layer_dense(units = 5, activation = 'relu', input_shape = c(13)) %>%
         layer_dense(units = 1)


#Model Compilation
model %>% compile(loss = 'mse',
                  optimizer = 'rmsprop',
                  metrics = 'mae')

#Model Fitting
mymodel <- model %>%
  fit(training, trainingtarget,
      epochs = 200,
      batch_size = 32,
      validation_split = 0.2)


#Prediction
model %>% evaluate(test, testtarget)
```

```
##      loss       mae
## 42.972755  4.620593
```

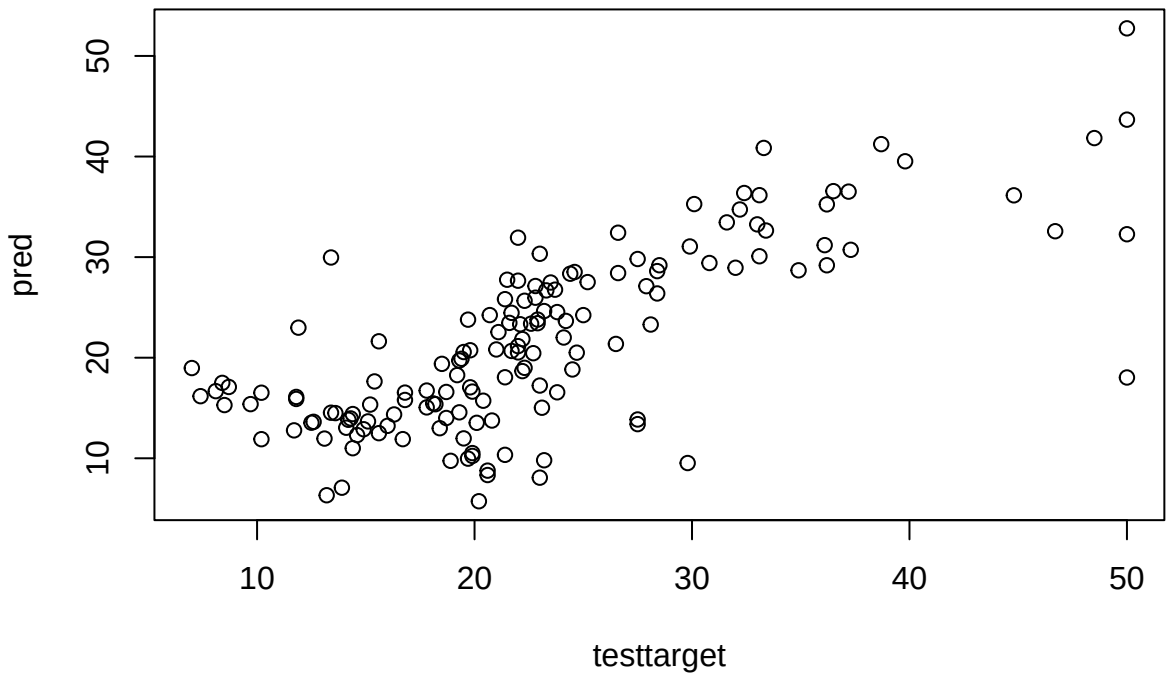
```
pred <- model %>% predict(test)

#MSE and MAE (The same ones as above)
mean((testtarget-pred)^2); mean(abs(testtarget-pred))
```

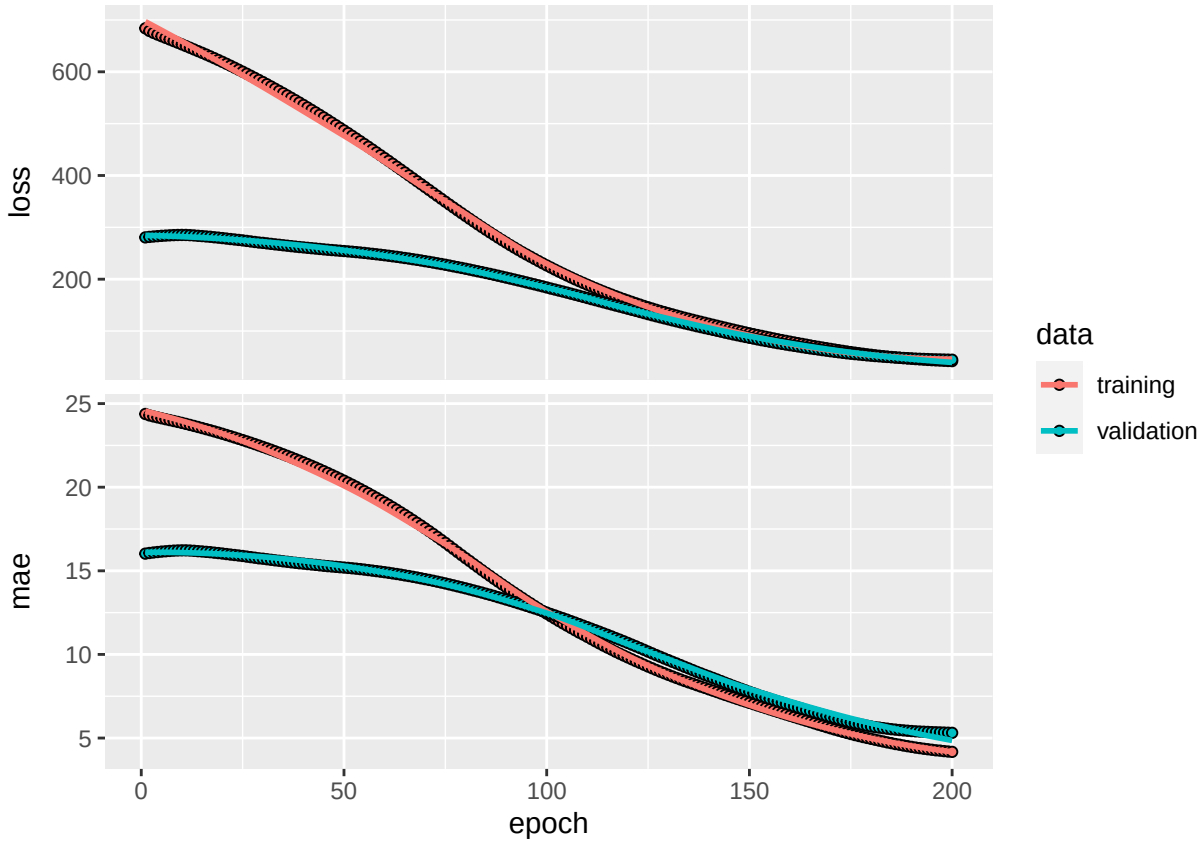
```
## [1] 42.97276
```

```
## [1] 4.620593
```

```
#Scatter Plot Original vs Predicted
plot(testtarget, pred)
```




```
#Plot model error for each epoch
plot(mymodel)
```


loss
600
400
200
mae
25
20
15
10
5
0
50
100
150
200
epoch
data
training
validation

Task 6

What does activation <- nn_relu() mean? What does dropout <- nn_dropout(0.4) mean?

nn_relu() is specifying the rectified linear unit function as our activation function of choice. This activation function is then used in our neural network.

ReLU activation function is defined as

$$g(z) = \begin{cases} z, & z > 0 \\ 0, & z <= 0 \end{cases}$$

nn_dropout(0.4) is randomly zeroing some of the units with a probability of 40%. This regularization can be applied to avoid overfitting

Task 7

Experiment with adding a second hidden layer. Does it improve the test performance?

```
#Model Creation
model2 <- keras_model_sequential()
model2 %>%
  layer_dense(units = 5, activation = 'relu', input_shape = c(13)) %>%
  layer_dropout(rate=0.3)  %>%
  layer_dense(units = 10, activation = 'relu')  %>%
  layer_dropout(rate=0.2)  %>%
  layer_dense(units = 1)


#Model Compilation
model2 %>% compile(loss = 'mse',
                   optimizer = 'rmsprop',
                   metrics = 'mae')

#Model Fitting
mymodel2 <- model2 %>%
  fit(training, trainingtarget,
      epochs = 200,
      batch_size = 32,
      validation_split = 0.2)


#Prediction
model2 %>% evaluate(test, testtarget)
```

```
##      loss       mae
## 29.806849  3.927519
```

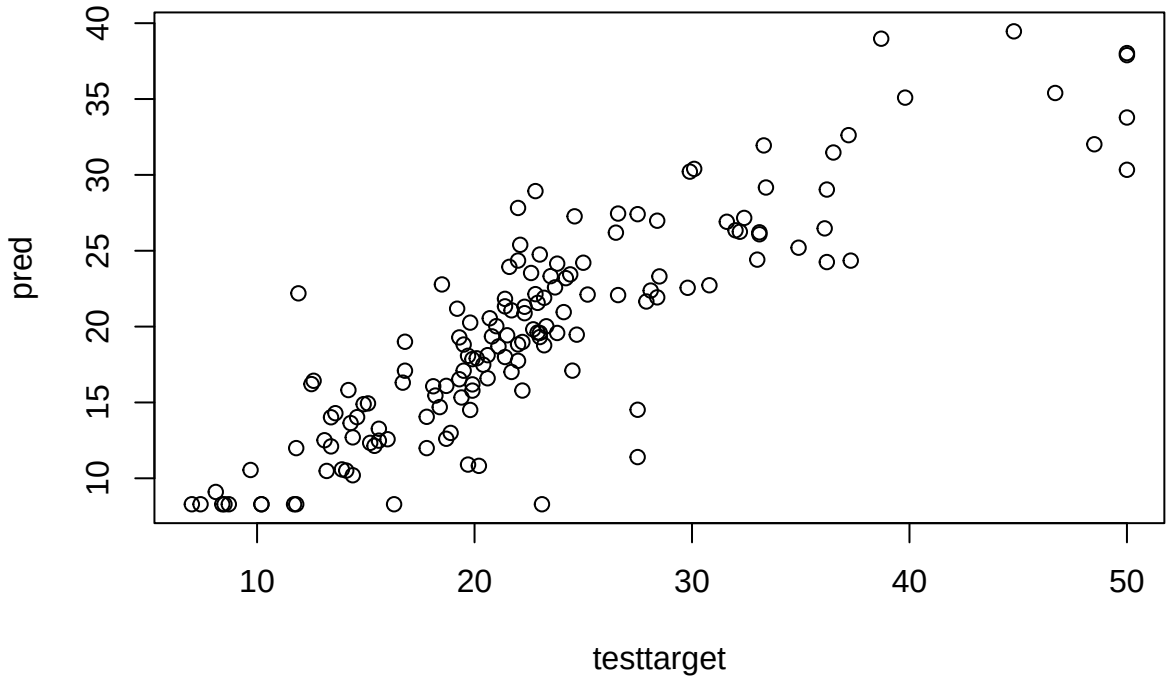
```
pred <- model2 %>% predict(test)

#MSE and MAE (The same ones as above)
mean((testtarget-pred)^2); mean(abs(testtarget-pred))
```

```
## [1] 29.80685
```

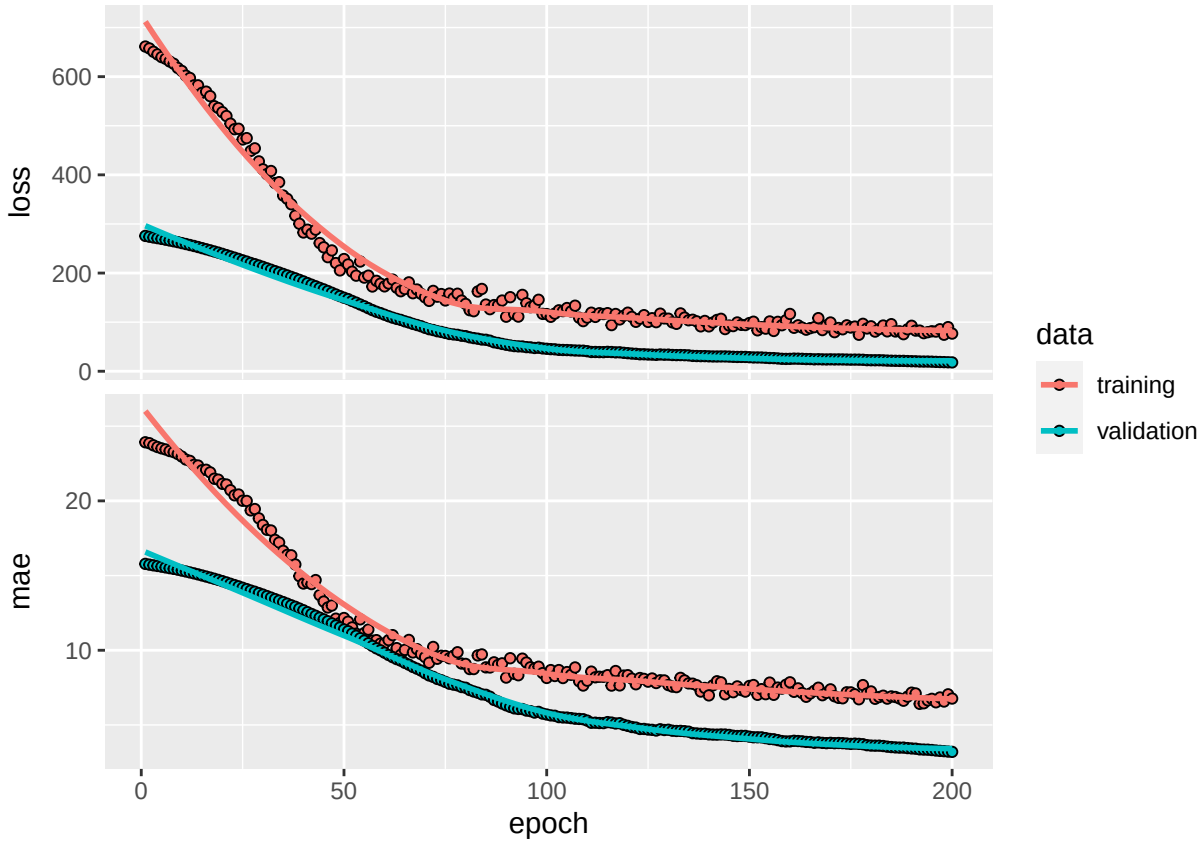
```
## [1] 3.927519
```

```
#Scatter plot Original vs Predicted
plot(testtarget, pred)
```




```
#Plot model error for each epoch
plot(mymodel2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



A second layer with 10 nodes seems to improve prediction performance. However, I also tested a model with 5 nodes in the second layer, which seemed to perform worse than the model with a single layer.

Task 8

Select a final neural network. Compare the MAE and MSE on the test set with linear regression with regularisation.

```
#Neural network with one hidden layer
model %>% evaluate(test, testtarget)
```

```
##      loss       mae
## 42.972755  4.620593
```

```
pred <- model %>% predict(test)

#MSE and MAE (The same ones as above)
NN1MSE = mean((testtarget-pred)^2)
NN1MAE = mean(abs(testtarget-pred))


#Neural network with two hidden layers
model2 %>% evaluate(test, testtarget)
```

```
##      loss       mae
## 29.806849  3.927519
```

```
pred <- model2 %>% predict(test)

#MSE and MAE (The same ones as above)
NN2MSE = mean((testtarget-pred)^2)
NN2MAE = mean(abs(testtarget-pred))


#Use function glmnet (from library glmnet) to perform linear regression with Lasso regularisation
#Alpha=1 means lasso.
#We use it on the same training data as was used for the neural network models
lassoModel = glmnet(x=training, y=trainingtarget, alpha=1, standardize=TRUE)

#Use 10-fold cross-validation (on the training data) for finding optimal lambda
lassoCV=cv.glmnet(x=training, y=trainingtarget, alpha=1, standardize=TRUE, nfolds=10)

#Test lasso on the same test data as used for the neural network models
lassoPred=predict(lassoModel, s=lassoCV$lambda.min, newx=test)

#MSE and MAE
lassoMSE = mean((lassoPred-testtarget)^2)
lassoMAE = mean(abs(lassoPred-testtarget))


#For nicer output (printing)
comparisonMatrix = matrix(c(NN1MSE, NN1MAE,
                            NN2MSE, NN2MAE,
                            lassoMSE, lassoMAE),
                          ncol=2,
```

```
                          byrow=TRUE)
comparisonMatrix = round(comparisonMatrix, 4)

colnames(comparisonMatrix) = c("MSE", "MAE")
rownames(comparisonMatrix) = c("Neural network with one hidden layer",
                               "Neural network with two hidden layers",
                               "Linear regression with Lasso regularisation")
print(comparisonMatrix)
```

```
##                                                MSE    MAE
## Neural network with one hidden layer        42.9728 4.6206
## Neural network with two hidden layers       29.8069 3.9275
## Linear regression with Lasso regularisation 21.9208 3.5699
```

Lasso seems to have the best prediction performance. The performance of the neural network might improve if different parameters are used E.g. I noticed that using a lower droupout rate and more neurons in the second layer improved performance. The number of epochs for training is another factor.