ANN in R

Dr Sebnem Er

2021-05-20

Contents

1 Introduction **5**

2 Neuralnet Package **7**

2.1 Example 1 - Boston 7

2.2 Example 2 - Iris Dataset 12

3 H2O **19**

3.1 Initialise H2O Connection 19

3.2 Data in H2o format 20

4 Keras **33**

4.1 Installing and Calling the Keras Package 33

4.2 Keras for Regression Problems 34

4.3 Keras for Classification Problems 38

4.4 Overfitting 42

Chapter 1

Introduction

In this tutorial you will be introduced to several R packages and how to use some of them such as *neuralnet* , *keras* and *h2o* .

In any of these packages the order of NN application is standard:

1 - Preprocess your data:

- Standardize or normalize your data (scale or min-max)
- Missing values, outliers
- Training/Validation and Test set splits

2 - Construct your NN model, ie. number of neurons, number of layers, learning rate, regularization etc. (these are your hyperparameters)

3 - Fit your model

4 - Predict your Y variable for both training and validation sets.

5 - Extract the training and validation errors, visualize these for different number of hyperparameters (fine tuning)

6 - Choose the parameters based on the smallest validation error.

7 - Assess the performance of your predicted model on the test set. Remember the test set error is not for decision making!

Chapter 2

Neuralnet Package

```
#install.packages("neuralnet")
rm(list=ls())
library(neuralnet)
```

2.1 Example 1 - Boston

```
library(MASS)
rm(list=ls())
data(Boston)
```

2.1.1 Scale, Training and Test Datasets

```
Boston.st = scale(Boston)
set.seed(1)
index <- sample(1:nrow(Boston.st),round(0.8*nrow(Boston.st)))
trainBoston.st <- Boston.st[index,]
testBoston.st <- Boston.st[-index,]
```

2.1.2 Build model

```r
set.seed(1)
neuralnetmodel = neuralnet(medv~crim+nox, data = trainBoston.st,
                           linear.output = TRUE,
                           act.fct = "logistic",
                           hidden = c(2))
```

```r
plot(neuralnetmodel)
```

```r
neuralnetmodel$weights
```

```
## [[1]]
## [[1]][[1]]
##             [,1]      [,2]
## [1,]  -38.34128  5.724901
## [2,]   20.11112 -3.802220
## [3,] -122.96637 -2.323184
##
## [[1]][[2]]
##             [,1]
## [1,] -1.4262287
## [2,]  0.6035351
## [3,]  1.3660896
```

```r
neuralnetmodel$err.fct
```

```
## function (x, y)
## {
##     1/2 * (y - x)^2
## }
## <bytecode: 0x000000001c462b10>
## <environment: 0x0000000021e72378>
## attr(,"type")
## [1] "sse"
```

```r
neuralnetmodel$act.fct
```

```
## function (x)
## {
##     1/(1 + exp(-x))
## }
## <bytecode: 0x000000001c45c288>
## <environment: 0x0000000021e753a8>
## attr(,"type")
## [1] "logistic"
```

2.1.3 Get the predictions train set

```
train.st.y.predict = predict(neuralnetmodel,trainBoston.st)
head(train.st.y.predict, 3)
```

```
##            [,1]
## 505 -0.06150515
## 324  0.54309902
## 167 -0.06612987
```

```
1/2*(sum((trainBoston.st[,"medv"]-as.numeric(train.st.y.predict))^2))
```

```
## [1] 153.874
```

```
#[1] 153.874
```

```
average.error.train = 1/(405)*(sum((trainBoston.st[,"medv"]-as.numeric(train.st.y.predict))^2))
average.error.train
```

```
## [1] 0.7598714
```

```
153.874*2/405
```

```
## [1] 0.7598716
```

2.1.4 Get the predictions test set

```
test.st.y.predict = predict(neuralnetmodel,testBoston.st)
head(test.st.y.predict, 3)
```

```
##          [,1]
## 6  0.54326440
## 7 -0.06064561
## 9 -0.06067375
```

```
1/2*(sum((testBoston.st[,"medv"]-as.numeric(test.st.y.predict))^2))
```

```
## [1] 22.56024
```

```
#[1] 22.56024
```

```
average.error.test = 1/(101)*(sum((testBoston.st[,"medv"]-as.numeric(test.st.y.predict)
average.error.test
```

```
## [1] 0.4467374
```

2.1.5 Forward propagation

```
head(trainBoston.st[,c("crim","nox","medv")])
```

```
##            crim        nox        medv
## 505 -0.40736095  0.1579678 -0.05793197
## 324 -0.38709366 -0.5324154 -0.43848654
## 167 -0.18640065  0.4341211  2.98650460
## 129 -0.38226778  0.5980871 -0.49285148
## 418  2.59570533  1.0727255 -1.31919854
## 471  0.08548074  0.2183763 -0.28626471
```

```
trainBoston.st[1,c("crim","nox","medv")]
```

```
##        crim         nox        medv
## -0.40736095  0.15796779 -0.05793197
```

```
Z1_2 = sum(neuralnetmodel$weights[[1]][[1]][,1]*c(1,trainBoston.st[1,c("crim","nox")]))
Z1_2
```

```
## [1] -65.95849
```

```
a1_2 = 1/(1+exp(-Z1_2))
a1_2
```

```
## [1] 2.26251e-29
```

```
Z2_2 = sum(neuralnetmodel$weights[[1]][[1]][,2]*c(1,trainBoston.st[1,c("crim","nox")]))
Z2_2
```

```
## [1] 6.906789
```

```
a2_2 = 1/(1+exp(-Z2_2))
a2_2
```

```
## [1] 0.999
```

```
Z1_3 = sum(neuralnetmodel$weights[[1]][[2]][,1]*c(1,a1_2,a2_2))

a1_3=Z1_3
a1_3
```

```
## [1] -0.06150515
```

2.1.6 Model Tuning

Although these seem like good results this may simply be a result of the subseted training and testing data so it is important to test the model performance further. In this example I will perform k-fold cross validation using 10 folds (10 fold cross validation)

get the validation indeces using the createFolds function provided by the

caret package

```
#install.packages("caret")
library(caret)
set.seed(1)
folds <- createFolds(trainBoston.st[,"medv"], k = 10)
#results is a vector that will contain the average sum error square for each
# of the network trainings for the validation set.
```

```
#errors.cv.number.of.neurons <- c()

#for (hidden in 1:2){
errors.cv.validation <- c()
for (fld in folds){

  #train the network (note I have subsetted out the indeces in the validation set)
  set.seed(1)
  neuralnetmodel <- neuralnet(medv~crim+nox, data = trainBoston.st[-fld,],
                  linear.output = TRUE,
                  act.fct = "logistic",
                  hidden = c(2))

  #get the predictions from the network for the validation set
```

```
  yhat.fld <- predict(neuralnetmodel, trainBoston.st[fld,])

  errorsq.fld = (trainBoston.st[fld,"medv"]-
                                          as.numeric(yhat.fld))^2

  errors.cv.validation <- c(errors.cv.validation, errorsq.fld)
}

#errors.cv.number.of.neurons[hidden] =

sum(errors.cv.validation)/dim(trainBoston.st)[1]
```

```
## [1] 0.7712774
```

2.2 Example 2 - Iris Dataset

```
rm(list=ls())

set.seed(1)
index <- sample(1:nrow(iris),round(0.8*nrow(iris)))
trainiris <- iris[index,]
testiris <- iris[-index,]
```

```
set.seed(1)
neuralnetmodel = neuralnet(Species~Petal.Length+Petal.Width, data = trainiris,
                          linear.output = FALSE,
                          act.fct = "logistic",
                          hidden = c(2), err.fct = "ce")
# stepmax = 1000000, threshold = 0.001, rep = 10 values can be changed as well.
```

```
plot(neuralnetmodel)
```

```
neuralnetmodel$weights
```

```
## [[1]]
## [[1]][[1]]
##           [,1]       [,2]
## [1,]  9.189152 -124.80564
## [2,] -1.436100   33.13635
## [3,] -1.300787   52.19249
##
```

```
## [[1]][[2]]
##             [,1]      [,2]       [,3]
## [1,]   -2.83673 -38.59970   4.022241
## [2,]   39.67766  28.62683 -28.834230
## [3,] -113.70312  24.75467   9.921314
```

```r
neuralnetmodel$err.fct
```

```
## function (x, y)
## {
##     -(y * log(x) + (1 - y) * log(1 - x))
## }
## <bytecode: 0x000000001c4622f8>
## <environment: 0x000000005f2a5bf8>
## attr(,"type")
## [1] "ce"
```

```r
neuralnetmodel$act.fct
```

```
## function (x)
## {
##     1/(1 + exp(-x))
## }
## <bytecode: 0x000000001c45c288>
## <environment: 0x000000005f2a5760>
## attr(,"type")
## [1] "logistic"
```

```r
train.iris.predict = predict(neuralnetmodel, trainiris)
head(round(train.iris.predict,3), 3)
```

```
##     [,1] [,2] [,3]
## 68     0    1    0
## 129    0    0    1
## 43     1    0    0
```

```r
maxprobability <- apply(train.iris.predict, 1, which.max)
train.iris.predict.class <- c('setosa', 'versicolor', 'virginica')[maxprobability]
head(train.iris.predict.class)
```

```
## [1] "versicolor" "virginica"  "setosa"     "setosa"     "versicolor"
## [6] "versicolor"
```

```
table(train.iris.predict.class, trainiris$Species)
```

```
##
## train.iris.predict.class setosa versicolor virginica
##               setosa         39          0         0
##               versicolor      0         36         2
##               virginica       0          2        41
```

```
mean(train.iris.predict.class== trainiris$Species)*100
```

```
## [1] 96.66667
```

```
test.iris.predict = predict(neuralnetmodel, testiris)
head(round(train.iris.predict,3), 3)
```

```
##     [,1] [,2] [,3]
## 68     0    1    0
## 129    0    0    1
## 43     1    0    0
```

```
maxprobability <- apply(test.iris.predict, 1, which.max)
test.iris.predict.class <- c('setosa', 'versicolor', 'virginica')[maxprobability]
head(test.iris.predict.class)
```

```
## [1] "setosa" "setosa" "setosa" "setosa" "setosa" "setosa"
```

```
table(test.iris.predict.class, testiris$Species)
```

```
##
## test.iris.predict.class setosa versicolor virginica
##              setosa         11          0         0
##              versicolor      0         12         2
##              virginica       0          0         5
```

```
mean(test.iris.predict.class== testiris$Species)*100
```

```
## [1] 93.33333
```

2.2.1 Model Tuning

Get the validation indices using the createFolds function provided by the caret package

```
set.seed(1)
folds <- createFolds(iris$Species, k = 10)
#results is a vector that will contain the accuracy for each of the network trainings and testing
results <- c()
for (fld in folds){
  #train the network
  set.seed(1)
  nn <- neuralnet(Species~Petal.Length+Petal.Width, data = iris[-fld,],
                  hidden = c(2), err.fct = "ce", act.fct = "logistic",
                  linear.output = FALSE, stepmax = 1000000)

  #get the classifications from the network
  classes <- predict(nn, iris[fld,])
  maxprobability <- apply(classes, 1, which.max)
  valid.iris.predict.class <- c('setosa', 'versicolor', 'virginica')[maxprobability]

  results <- c(results, valid.iris.predict.class== iris[fld,"Species"])
}

results
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [37]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [73]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [85]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [97]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## [109]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [121]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [145]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
sum(results)/length(results)
```

```
## [1] 0.96
```

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter 4.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

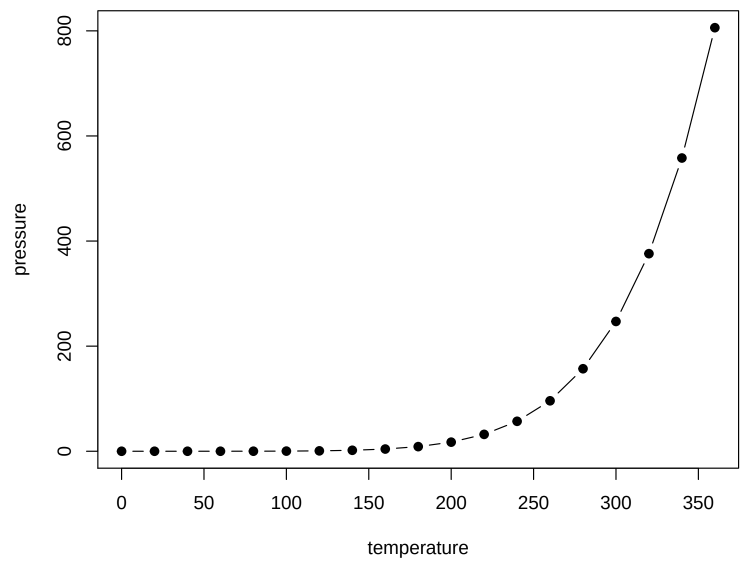

Figure 2.1: Here is a nice figure!

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 2.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 2.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2020) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

Table 2.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 3

H2O

H2O is probably the easiest to learn and use.

```
## Load all packages first
library(h2o)
library(caret)
library(mlbench)
library(ggplot2)
library(reshape2)
library(DEEPR)

# http://blog.revolutionanalytics.com/2014/04/a-dive-into-h2o.html
# https://discuss.analyticsvidhya.com/t/script-in-h2o-in-r-to-get-you-into-top-30-percentile-for-
```

3.1 Initialise H2O Connection

```
## Start a local H2O cluster directly from R
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,min_mem_size = "3g")
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 1 minutes
##     H2O cluster timezone:       Africa/Johannesburg
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.32.0.1
##     H2O cluster version age:    7 months and 11 days !!!
```

```
##     H2O cluster name:           H2O_started_from_R_01438475_sup231
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.89 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, C
##     R Version:                  R version 4.0.1 (2020-06-06)
```

```
## Warning in h2o.clusterInfo():
## Your H2O cluster version is too old (7 months and 11 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

```
#Get help
?h2o.deeplearning
```

3.2 Data in H2o format

```
# iris data ####
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
```

```
## 17          5.4        3.9         1.3         0.4     setosa
## 18          5.1        3.5         1.4         0.3     setosa
## 19          5.7        3.8         1.7         0.3     setosa
## 20          5.1        3.8         1.5         0.3     setosa
## 21          5.4        3.4         1.7         0.2     setosa
## 22          5.1        3.7         1.5         0.4     setosa
## 23          4.6        3.6         1.0         0.2     setosa
## 24          5.1        3.3         1.7         0.5     setosa
## 25          4.8        3.4         1.9         0.2     setosa
## 26          5.0        3.0         1.6         0.2     setosa
## 27          5.0        3.4         1.6         0.4     setosa
## 28          5.2        3.5         1.5         0.2     setosa
## 29          5.2        3.4         1.4         0.2     setosa
## 30          4.7        3.2         1.6         0.2     setosa
## 31          4.8        3.1         1.6         0.2     setosa
## 32          5.4        3.4         1.5         0.4     setosa
## 33          5.2        4.1         1.5         0.1     setosa
## 34          5.5        4.2         1.4         0.2     setosa
## 35          4.9        3.1         1.5         0.2     setosa
## 36          5.0        3.2         1.2         0.2     setosa
## 37          5.5        3.5         1.3         0.2     setosa
## 38          4.9        3.6         1.4         0.1     setosa
## 39          4.4        3.0         1.3         0.2     setosa
## 40          5.1        3.4         1.5         0.2     setosa
## 41          5.0        3.5         1.3         0.3     setosa
## 42          4.5        2.3         1.3         0.3     setosa
## 43          4.4        3.2         1.3         0.2     setosa
## 44          5.0        3.5         1.6         0.6     setosa
## 45          5.1        3.8         1.9         0.4     setosa
## 46          4.8        3.0         1.4         0.3     setosa
## 47          5.1        3.8         1.6         0.2     setosa
## 48          4.6        3.2         1.4         0.2     setosa
## 49          5.3        3.7         1.5         0.2     setosa
## 50          5.0        3.3         1.4         0.2     setosa
## 51          7.0        3.2         4.7         1.4 versicolor
## 52          6.4        3.2         4.5         1.5 versicolor
## 53          6.9        3.1         4.9         1.5 versicolor
## 54          5.5        2.3         4.0         1.3 versicolor
## 55          6.5        2.8         4.6         1.5 versicolor
## 56          5.7        2.8         4.5         1.3 versicolor
## 57          6.3        3.3         4.7         1.6 versicolor
## 58          4.9        2.4         3.3         1.0 versicolor
## 59          6.6        2.9         4.6         1.3 versicolor
## 60          5.2        2.7         3.9         1.4 versicolor
## 61          5.0        2.0         3.5         1.0 versicolor
## 62          5.9        3.0         4.2         1.5 versicolor
```

```
## 63           6.0         2.2          4.0         1.0 versicolor
## 64           6.1         2.9          4.7         1.4 versicolor
## 65           5.6         2.9          3.6         1.3 versicolor
## 66           6.7         3.1          4.4         1.4 versicolor
## 67           5.6         3.0          4.5         1.5 versicolor
## 68           5.8         2.7          4.1         1.0 versicolor
## 69           6.2         2.2          4.5         1.5 versicolor
## 70           5.6         2.5          3.9         1.1 versicolor
## 71           5.9         3.2          4.8         1.8 versicolor
## 72           6.1         2.8          4.0         1.3 versicolor
## 73           6.3         2.5          4.9         1.5 versicolor
## 74           6.1         2.8          4.7         1.2 versicolor
## 75           6.4         2.9          4.3         1.3 versicolor
## 76           6.6         3.0          4.4         1.4 versicolor
## 77           6.8         2.8          4.8         1.4 versicolor
## 78           6.7         3.0          5.0         1.7 versicolor
## 79           6.0         2.9          4.5         1.5 versicolor
## 80           5.7         2.6          3.5         1.0 versicolor
## 81           5.5         2.4          3.8         1.1 versicolor
## 82           5.5         2.4          3.7         1.0 versicolor
## 83           5.8         2.7          3.9         1.2 versicolor
## 84           6.0         2.7          5.1         1.6 versicolor
## 85           5.4         3.0          4.5         1.5 versicolor
## 86           6.0         3.4          4.5         1.6 versicolor
## 87           6.7         3.1          4.7         1.5 versicolor
## 88           6.3         2.3          4.4         1.3 versicolor
## 89           5.6         3.0          4.1         1.3 versicolor
## 90           5.5         2.5          4.0         1.3 versicolor
## 91           5.5         2.6          4.4         1.2 versicolor
## 92           6.1         3.0          4.6         1.4 versicolor
## 93           5.8         2.6          4.0         1.2 versicolor
## 94           5.0         2.3          3.3         1.0 versicolor
## 95           5.6         2.7          4.2         1.3 versicolor
## 96           5.7         3.0          4.2         1.2 versicolor
## 97           5.7         2.9          4.2         1.3 versicolor
## 98           6.2         2.9          4.3         1.3 versicolor
## 99           5.1         2.5          3.0         1.1 versicolor
## 100          5.7         2.8          4.1         1.3 versicolor
## 101          6.3         3.3          6.0         2.5  virginica
## 102          5.8         2.7          5.1         1.9  virginica
## 103          7.1         3.0          5.9         2.1  virginica
## 104          6.3         2.9          5.6         1.8  virginica
## 105          6.5         3.0          5.8         2.2  virginica
## 106          7.6         3.0          6.6         2.1  virginica
## 107          4.9         2.5          4.5         1.7  virginica
## 108          7.3         2.9          6.3         1.8  virginica
```

```
## 109          6.7         2.5          5.8         1.8  virginica
## 110          7.2         3.6          6.1         2.5  virginica
## 111          6.5         3.2          5.1         2.0  virginica
## 112          6.4         2.7          5.3         1.9  virginica
## 113          6.8         3.0          5.5         2.1  virginica
## 114          5.7         2.5          5.0         2.0  virginica
## 115          5.8         2.8          5.1         2.4  virginica
## 116          6.4         3.2          5.3         2.3  virginica
## 117          6.5         3.0          5.5         1.8  virginica
## 118          7.7         3.8          6.7         2.2  virginica
## 119          7.7         2.6          6.9         2.3  virginica
## 120          6.0         2.2          5.0         1.5  virginica
## 121          6.9         3.2          5.7         2.3  virginica
## 122          5.6         2.8          4.9         2.0  virginica
## 123          7.7         2.8          6.7         2.0  virginica
## 124          6.3         2.7          4.9         1.8  virginica
## 125          6.7         3.3          5.7         2.1  virginica
## 126          7.2         3.2          6.0         1.8  virginica
## 127          6.2         2.8          4.8         1.8  virginica
## 128          6.1         3.0          4.9         1.8  virginica
## 129          6.4         2.8          5.6         2.1  virginica
## 130          7.2         3.0          5.8         1.6  virginica
## 131          7.4         2.8          6.1         1.9  virginica
## 132          7.9         3.8          6.4         2.0  virginica
## 133          6.4         2.8          5.6         2.2  virginica
## 134          6.3         2.8          5.1         1.5  virginica
## 135          6.1         2.6          5.6         1.4  virginica
## 136          7.7         3.0          6.1         2.3  virginica
## 137          6.3         3.4          5.6         2.4  virginica
## 138          6.4         3.1          5.5         1.8  virginica
## 139          6.0         3.0          4.8         1.8  virginica
## 140          6.9         3.1          5.4         2.1  virginica
## 141          6.7         3.1          5.6         2.4  virginica
## 142          6.9         3.1          5.1         2.3  virginica
## 143          5.8         2.7          5.1         1.9  virginica
## 144          6.8         3.2          5.9         2.3  virginica
## 145          6.7         3.3          5.7         2.5  virginica
## 146          6.7         3.0          5.2         2.3  virginica
## 147          6.3         2.5          5.0         1.9  virginica
## 148          6.5         3.0          5.2         2.0  virginica
## 149          6.2         3.4          5.4         2.3  virginica
## 150          5.9         3.0          5.1         1.8  virginica
```

```
index <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
irisTrain = iris[index,]
```

```
irisTest = iris[-index,]

iris.h2oTrain <- as.h2o(irisTrain)
```

```
## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
iris.h2oTest <- as.h2o(irisTest)
```

```
## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
iris.nn <- h2o.deeplearning(x = 1:4 ,
                            y = 5,
                            training_frame = iris.h2oTrain, # data in H2O format
                            validation_frame = iris.h2oTest,
                            activation = "Tanh",
                            hidden = c(5), # one layer of 5 nodes
                            l1 = 1e-5,
                            epochs = 100, variable_importances = TRUE) # max. no. of e
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
iris.nn.cv <- h2o.deeplearning(x = 1:4 ,
                               y = 5,
                               training_frame = iris.h2oTrain, # data in H2O format
                               validation_frame = iris.h2oTest,
                               activation = "Tanh",
                               hidden = c(5), # one layer of 5 nodes
                               l1 = 1e-5,
                               nfolds = 5,
                               epochs = 100) # max. no. of epochs
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
iris.nn@parameters
```

```
## $model_id
## [1] "DeepLearning_model_R_1621504187964_9"
##
## $training_frame
## [1] "irisTrain_sid_8eef_7"
##
## $validation_frame
## [1] "irisTest_sid_8eef_9"
##
## $activation
## [1] "Tanh"
##
## $hidden
## [1] 5
##
## $epochs
## [1] 100
##
## $seed
## [1] "-7001910658379984480"
##
## $l1
## [1] 1e-05
##
## $distribution
## [1] "multinomial"
```

```
##
## $stopping_metric
## [1] "logloss"
##
## $categorical_encoding
## [1] "OneHotInternal"
##
## $x
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
##
## $y
## [1] "Species"
```

```
h2o.performance(iris.nn, train = TRUE)
```

```
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("irisTrain_sid_8eef_7")`
## MSE: (Extract with `h2o.mse`) 0.0321675
## RMSE: (Extract with `h2o.rmse`) 0.179353
## Logloss: (Extract with `h2o.logloss`) 0.1239403
## Mean Per-Class Error: 0.06666667
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##            setosa versicolor virginica  Error     Rate
## setosa         25          0         0 0.0000 = 0 / 25
## versicolor      0         24         1 0.0400 = 1 / 25
## virginica       0          4        21 0.1600 = 4 / 25
## Totals         25         28        22 0.0667 = 5 / 75
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-3 Hit Ratios:
##   k hit_ratio
## 1 1  0.933333
## 2 2  1.000000
## 3 3  1.000000
```

```
h2o.mse(iris.nn, train = TRUE)
```

```
## [1] 0.0321675
```

```
h2o.mse(iris.nn, valid = TRUE)
```

```
## [1] 0.05660571
```

```
h2o.mse(iris.nn.cv, xval = TRUE)
```

```
## [1] 0.04916102
```

```
h2o.varimp(iris.nn)
```

```
## Variable Importances:
##       variable relative_importance scaled_importance percentage
## 1 Petal.Length            1.000000          1.000000   0.357745
## 2  Sepal.Width            0.672481          0.672481   0.240577
## 3  Petal.Width            0.637729          0.637729   0.228144
## 4 Sepal.Length            0.485079          0.485079   0.173535
```

```
# now make a prediction
predictionsTrain <- h2o.predict(iris.nn, iris.h2oTrain)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
predictionsTrain
```

```
##   predict    setosa  versicolor    virginica
## 1  setosa 0.9189597 0.078154570 2.885716e-03
## 2  setosa 0.9917346 0.008200951 6.440809e-05
## 3  setosa 0.9921608 0.007773779 6.539636e-05
## 4  setosa 0.9917609 0.008190904 4.815380e-05
## 5  setosa 0.9938607 0.006100744 3.855432e-05
## 6  setosa 0.9931896 0.006765612 4.476855e-05
##
## [75 rows x 4 columns]
```

```
predictionsTest <- h2o.predict(iris.nn, iris.h2oTest)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
predictionsTest
```

```
##   predict    setosa  versicolor    virginica
## 1  setosa 0.9915620 0.008376958 6.100055e-05
## 2  setosa 0.9929519 0.006970218 7.785587e-05
## 3  setosa 0.9925826 0.007323164 9.424385e-05
## 4  setosa 0.9927470 0.007202587 5.044060e-05
## 5  setosa 0.9938037 0.006142305 5.402424e-05
## 6  setosa 0.9912707 0.008660765 6.855310e-05
##
## [75 rows x 4 columns]
```

```
yhatTrain = as.factor(as.matrix(predictionsTrain$predict))
confusionMatrix(yhatTrain, irisTrain$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         25          0         0
##   versicolor      0         24         4
##   virginica       0          1        21
##
## Overall Statistics
##
##                Accuracy : 0.9333
##                  95% CI : (0.8512, 0.978)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.9600           0.8400
## Specificity                 1.0000            0.9200           0.9800
## Pos Pred Value              1.0000            0.8571           0.9545
## Neg Pred Value              1.0000            0.9787           0.9245
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3200           0.2800
## Detection Prevalence        0.3333            0.3733           0.2933
## Balanced Accuracy           1.0000            0.9400           0.9100
```

```
yhatTest = as.factor(as.matrix(predictionsTest$predict))
confusionMatrix(yhatTest, irisTest$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         25          0         0
##   versicolor      0         25         5
##   virginica       0          0        20
##
## Overall Statistics
##
##                Accuracy : 0.9333
##                  95% CI : (0.8512, 0.978)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.8000
## Specificity                 1.0000            0.9000           1.0000
## Pos Pred Value              1.0000            0.8333           1.0000
## Neg Pred Value              1.0000            1.0000           0.9091
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3333           0.2667
## Detection Prevalence        0.3333            0.4000           0.2667
## Balanced Accuracy           1.0000            0.9500           0.9000
```

3.2.1 Grid Search for Complexity

https://h2o-release.s3.amazonaws.com/h2o/master/3190/docs-website/h2o-docs/booklets/DeepLearning_Vignette.pdf

```
hidden_opt <- list(c(1), c(2), c(3), 4,5,6,7,8,9,10, c(3,4),c(4,4), c(5,4), c(6,4))
hyper_params <- list(hidden = hidden_opt)
model_grid <- h2o.grid("deeplearning",
  hyper_params = hyper_params,
  x = 1:4 ,
  y = 5,
  training_frame = iris.h2oTrain, # data in H2O format
  validation_frame = iris.h2oTest,
  activation = "Tanh",
  seed = 1, reproducible = TRUE, nfolds = 5
  )
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
model_grid
```

```
## H2O Grid Details
## ================
##
## Grid ID: Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11
## Used hyper parameters:
##   -  hidden
## Number of models: 14
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##    hidden
## 1  [4, 4]
## 2  [5, 4]
## 3  [6, 4]
## 4    [10]
## 5     [8]
## 6     [9]
## 7     [4]
## 8     [5]
## 9     [7]
```

```
## 10 [3, 4]
## 11    [1]
## 12    [6]
## 13    [3]
## 14    [2]
##                                                                    model_ids
## 1  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_12
## 2  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_13
## 3  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_14
## 4  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_10
## 5   Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_8
## 6   Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_9
## 7   Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_4
## 8   Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_5
## 9   Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_7
## 10 Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_11
## 11  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_1
## 12  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_6
## 13  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_3
## 14  Grid_DeepLearning_irisTrain_sid_8eef_7_model_R_1621504187964_11_model_2
##                logloss
## 1  0.30352114209833503
## 2  0.30840267615936423
## 3   0.3163368545946698
## 4   0.3522326361514222
## 5  0.38675768802641175
## 6   0.3888386663885484
## 7   0.4186838511287884
## 8  0.44581740710339185
## 9  0.44962730950467167
## 10  0.5538316590220661
## 11  0.5622986947815931
## 12  0.5715811429237513
## 13  0.5926435423066516
## 14  0.6974506769588046
```

```r
model1 = h2o.getModel(model_grid@model_ids[[1]])
h2o.mse(model1, xval = TRUE)
```

```
## [1] 0.0856723
```

Chapter 4

Keras

Developers: Daniel Falbel (Contributor, copyright holder, maintainer), JJ Allaire (Author, copyright holder), Francois Chollet (Author, copyright holder) etc.

https://keras.rstudio.com/ https://keras.rstudio.com/articles/sequential_model.html

The keras package uses the pipe operator to connect functions or operations together and the datasets need to be in matrix format.

4.1 Installing and Calling the Keras Package

```
#devtools::install_github("rstudio/keras")
library(keras)
library(tensorflow)
#install.packages("devtools")
#require(devtools)
#install_github("rstudio/reticulate")
#install_github("rstudio/tensorflow")
#install_github("rstudio/keras", force = TRUE)

# Install TensorFlow
#install_tensorflow()
library(keras)
#install_keras()
library(tensorflow)
#install_tensorflow()
```

```
library(reticulate)

#reticulate::py_available(initialize = TRUE)
#reticulate::py_versions_windows()
#reticulate::py_config()
#reticulate::py_discover_config("keras")
```

4.2 Keras for Regression Problems

4.2.1 Load the dataset, standardize

```
library(MASS)
rm(list=ls())
data(Boston)
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

```
# Standardize Your dataset
Boston.st = scale(Boston)

set.seed(1)
index <- sample(1:nrow(Boston.st),round(0.8*nrow(Boston.st)))
trainBoston.st <- Boston.st[index,]
testBoston.st <- Boston.st[-index,]
head(testBoston.st)
```

```
##           crim          zn       indus         chas          nox          rm
```

```
## 6  -0.4166314 -0.48724019 -1.3055857 -0.2723291 -0.8344581  0.2068916
## 7  -0.4098372  0.04872402 -0.4761823 -0.2723291 -0.2648919 -0.3880270
## 9  -0.3955433  0.04872402 -0.4761823 -0.2723291 -0.2648919 -0.9302853
## 10 -0.4003331  0.04872402 -0.4761823 -0.2723291 -0.2648919 -0.3994130
## 11 -0.3939564  0.04872402 -0.4761823 -0.2723291 -0.2648919  0.1314594
## 12 -0.4064448  0.04872402 -0.4761823 -0.2723291 -0.2648919 -0.3922967
##            age       dis        rad       tax    ptratio     black       lstat
## 6  -0.35080997 1.0766711 -0.7521778 -1.105022  0.1129203 0.4101651 -1.04229087
## 7  -0.07015919 0.8384142 -0.5224844 -0.576948 -1.5037485 0.4263763 -0.03123671
## 9   1.11638970 1.0861216 -0.5224844 -0.576948 -1.5037485 0.3281233  2.41937935
## 10  0.61548134 1.3283202 -0.5224844 -0.576948 -1.5037485 0.3289995  0.62272769
## 11  0.91389483 1.2117800 -0.5224844 -0.576948 -1.5037485 0.3926395  1.09184562
## 12  0.50890509 1.1547920 -0.5224844 -0.576948 -1.5037485 0.4406159  0.08639286
##           medv
## 6   0.67055821
## 7   0.03992492
## 9  -0.65594629
## 10 -0.39499459
## 11 -0.81904111
## 12 -0.39499459
```

```
# The data needs to be in a matrix format and X variables and Y variables should be provided in t

x_train.st <- as.matrix(trainBoston.st[,-14])
y_train.st <- as.matrix(trainBoston.st[,14])

x_test.st <- as.matrix(testBoston.st[,-14])
y_test.st <- as.matrix(testBoston.st[,14])

dim(x_train.st)
```

```
## [1] 405  13
```

```
dim(y_train.st)
```

```
## [1] 405   1
```

```
dim(x_test.st)
```

```
## [1] 101  13
```

```
dim(y_test.st)
```

```
## [1] 101   1
```

In fact all this is available in Keras from the dataset_boston_housing() function so you can also use the following, remember you still might need to standardize your dataset:

```
#boston_housing <- dataset_boston_housing()

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

4.2.2 Initialize a sequential feed forward neural network

```
# Initialize an empty sequential model:
model.regression <- keras_model_sequential()
```

4.2.3 Define the structure of your model: layers and the activation functions

Add layers to the model, 1 Input, 1 Hidden and 1 Output Layer

Layers are defined within the layer_dense() functions

We don't need to define the Input Layer separately since it gets associated with the 1st Hidden layer and no activation occurs in the Input Layer.

Though we do need to specify the Output Layer because we need to define the activation function and the dimension.

```
model.regression %>%
  layer_dense(units = 3, activation = 'relu', input_shape = c(13)) %>%
  # 13 X variables + 1 constant = input layer has 14 neurons
  # The 14 neurons are fully connected to 3 neurons in the hidden layer.
  # 14*2 = 42 weights are optimized
  # The activation of the hidden layer neurons

  layer_dense(units = 1, activation = 'relu')
  # 3 neurons in the hidden layer + 1 constant = 4 neurons in total
  # The 4 neurons are fully connected to 1 neuron in the output layer
  # 4*1 = 4 weights to optimize
  # the activation of the output layer, since a regression problem, best is either rel
```

```
# Print a summary of a model
summary(model.regression) # in total 46 parameters will be optimized.
```

```
## Model: "sequential_2"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## dense_5 (Dense)                     (None, 3)                           42
## ________________________________________________________________________________
## dense_4 (Dense)                     (None, 1)                           4
## ================================================================================
## Total params: 46
## Trainable params: 46
## Non-trainable params: 0
## ________________________________________________________________________________
```

4.2.4 Define how these weights will be optimized

This is where the weights will be optimized by minimizing the "Mean Square Error" which is calculated by $\frac{\sum_{i=1}^{n}(y-\hat{y})^2}{n}$.

```
model.regression %>% compile(
  optimizer = optimizer_rmsprop(lr = 0.002),
  loss = 'mse'
)
```

4.2.5 Run the model without validation

```
model.regression %>% fit(x_train.st, y_train.st, epochs = 20, batch_size = 32)
```

All this could have been done at once since we are using a pipeline operator.

We can then evaluate the performance of the model using the test set which means we need to predict the Y values of the test set using this model:

```
predictions.y.train.st = model.regression %>% predict(x_train.st)
mse.train.st = sum((y_train.st - predictions.y.train.st)^2)/dim(x_train.st)[1]
mse.train.st
```

```
## [1] 0.5071529
```

```
predictions.y.test.st = model.regression %>% predict(x_test.st)
mse.test.st = sum((y_test.st - predictions.y.test.st)^2)/dim(x_test.st)[1]
mse.test.st
```

```
## [1] 0.5687869
```

```
# same as below:
model.regression %>% evaluate(x_train.st, y_train.st, batch_size=32, verbose = 1)
```

```
##      loss
## 0.5071529
```

```
model.regression %>% evaluate(x_test.st, y_test.st, batch_size=32, verbose = 1)
```

```
##      loss
## 0.5687869
```

4.2.6 Run the model with validation

```
# Fit the model
model.regression %>% fit(
     x_train.st,
     y_train.st,
     epochs = 20,
     batch_size = 32,
     validation_split = 0.2
 )
```

4.3 Keras for Classification Problems

For classification problems, the Y variable needs to be converted into dummy variables (one-hot-encoding), and the output layer activation function needs to be logistic function:

```
rm(list=ls())
iris$numericclasses = unclass(iris$Species)
set.seed(1)
index <- sample(1:nrow(iris),round(0.8*nrow(iris)))
trainiris <- iris[index,]
```

```
testiris <- iris[-index,]

# no need to standardize this dataset but we still need to convert to matrices
x_iristrain <- as.matrix(trainiris[,c(1:4)])
y_iristrain <- as.matrix(trainiris[,6])

# Convert labels to categorical one-hot encoding
y_iristrain.one_hot_labels <- to_categorical(y_iristrain, num_classes = 4)

x_iristest <- as.matrix(testiris[,c(1:4)])
y_iristest <- as.matrix(testiris[,6])

# Convert labels to categorical one-hot encoding
y_iristest.one_hot_labels <- to_categorical(y_iristest, num_classes = 4)
```

4.3.1 Initialize and define a sequential feed forward neural network

The only difference here is we defined all the components together, and remember the output will have 3 neurons with a softmax or logistic activation function defined.

```
model.classification <- keras_model_sequential() %>%
  layer_dense(units = 3, activation = 'relu', input_shape = c(4)) %>%
  layer_dense(units = 3, activation = 'softmax') %>%
  compile(loss = 'categorical_crossentropy',
          optimizer = 'rmsprop',
          metrics = c('accuracy'))

summary(model.classification)
```

```
## Model: "sequential_3"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense_7 (Dense)                     (None, 3)                       15
## ________________________________________________________________________________
## dense_6 (Dense)                     (None, 3)                       12
## ================================================================================
## Total params: 27
## Trainable params: 27
## Non-trainable params: 0
## ________________________________________________________________________________
```

In total 27 parameters will be optimized. (4variables+1 neurons in the input layer = 5 neurons, connected to a hidden layer that has 3 neurons, in total 15 weights. 3 neurons + 1 bias = 4 neurons in the hidden layer are fully connected to the output layer which has 3 neurons, 4*3 = 12 weights to optimize)

4.3.2 Run the model without validation

```
model.classification %>% fit(x_iristrain, y_iristrain.one_hot_labels[,-1], epochs = 20
```

We can then evaluate the performance of the model using the test set which means we need to predict the Y values of the test set using this model:

```
model.classification %>% evaluate(x_iristrain, y_iristrain.one_hot_labels[,-1], batch_s
```

```
##      loss  accuracy
## 0.8015296 0.6000000
```

```
model.classification %>% evaluate(x_iristest, y_iristest.one_hot_labels[,-1], batch_siz
```

```
##      loss  accuracy
## 0.7815567 0.6000000
```

4.3.3 Run the model with validation

```
# Fit the model
model.classification %>% fit(x_iristrain, y_iristrain.one_hot_labels[,-1],
     epochs = 20,
     batch_size = 32,
     validation_split = 0.2
 )
```

4.3.4 Evaluate the model

```
model.classification %>% evaluate(x_iristrain, y_iristrain.one_hot_labels[,-1], batch_s
```

```
##      loss  accuracy
## 0.7560373 0.5833333
```

```
model.classification %>% evaluate(x_iristest, y_iristest.one_hot_labels[,-1], batch_size=32, verb
```

```
##      loss  accuracy
## 0.7407854 0.5333334
```

4.3.5 Obtain the predictions:

```
iris.train.predicted <- model.classification %>% predict(x_iristrain)

head(iris.train.predicted)
```

```
##           [,1]       [,2]      [,3]
## [1,] 0.1769677 0.44190410 0.3811281
## [2,] 0.1067968 0.53687590 0.3563273
## [3,] 0.7009802 0.06986266 0.2291571
## [4,] 0.6802513 0.07839867 0.2413500
## [5,] 0.2019714 0.42321557 0.3748131
## [6,] 0.2362157 0.36166942 0.4021149
```

```
sum(iris.train.predicted[1,])
```

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

```
maxidx <- function(arr) {
  return(which(arr == max(arr)))
}

whichmax.train.predicted <- apply(iris.train.predicted, c(1), maxidx)

head(whichmax.train.predicted)
```

```
## [1] 2 2 1 1 2 3
```

```
prediction.train <- c('setosa', 'versicolor', 'virginica')[whichmax.train.predicted]
actual.train <- c('setosa', 'versicolor', 'virginica')[trainiris$Species]

table(prediction.train, actual.train)
```

```
##                 actual.train
## prediction.train setosa versicolor virginica
```

```
##       setosa          39           0          0
##       versicolor       0          29         41
##       virginica        0           9          2
```

```
accuracy.train = mean(prediction.train==actual.train)
accuracy.train
```

```
## [1] 0.5833333
```

4.4 Overfitting

https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_overfit_underfit/

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.20.