Stat. 654 Homework 1

Jiayi Liu

March 18, 2019

Homework 1:

(due Monday March 25, 2019) Read: Chapter 1 and Chapter 2 Problems: Install R and RStudio, if you do not have them installed. Install the R packages kera and tensorflow for use with a CPU. Run the cars Example. Run the concrete Example. Run the iris Example. Run the code in Chapter 2. A first look at a neural network

Run the cars Example.

Example: Compare Simple Linear Regresion to a single layer NN.

The **cars** dataset in R contains two variables stopping *speed* of cars in mph and *dist* in feet. Using speed to predict stopping distance, two models are fit. See the R code.

a. What function is used to normalize the data?
b. What percentage of the data is used for *training*? What percentage of the data is used for *testing*?
c. What is the fitted linear regression model?
d. What is the correlation between the linear regression predicted values and the values from the test data?
e. Sketch the NN model that is used to model stopping distance.
f. What kind of activation function was used in the ANN? Sketch a picture of what the activation function looks like.
g. What is the correlation between the ANN predicted values and the values from the test data?
h. Examine the scatterplot of speed by distance with the fitted models. Is the NN fitting a near linear function?
i. Which model would you use for prediction? Explain.

Answer:

Read in data and examine structure.

```
suppressMessages(library("tidyverse"))
```

```
## Warning: package 'tibble' was built under R version 3.5.3

## Warning: package 'purrr' was built under R version 3.5.3

## Warning: package 'dplyr' was built under R version 3.5.3

## Warning: package 'forcats' was built under R version 3.5.2
```

```
cars <- as.tibble(cars)
```

```
## Warning: `as.tibble()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
cars
```

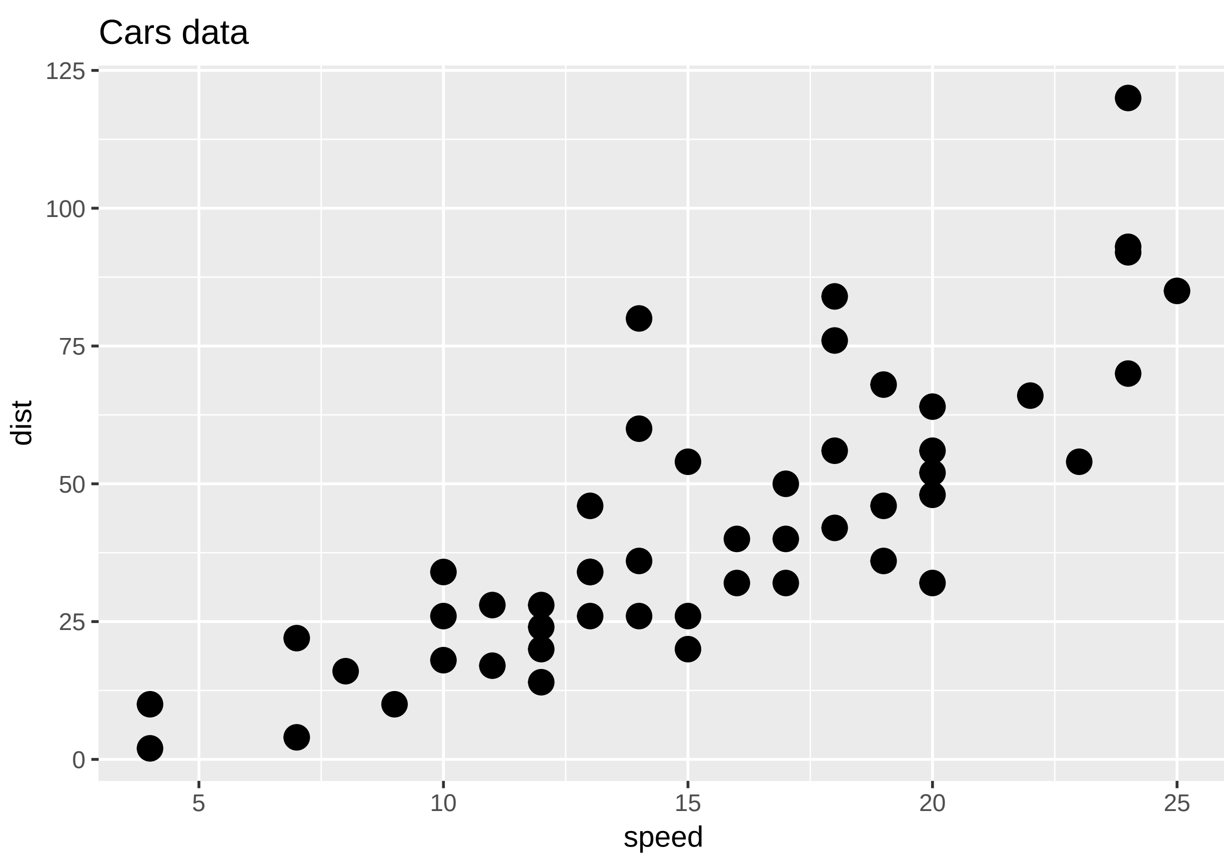
```
## # A tibble: 50 x 2
##    speed  dist
```

```
##    <dbl> <dbl>
##  1     4     2
##  2     4    10
##  3     7     4
##  4     7    22
##  5     8    16
##  6     9    10
##  7    10    18
##  8    10    26
##  9    10    34
## 10    11    17
## # ... with 40 more rows
```

```
str(cars)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

```
cars %>% ggplot(aes(x=speed, y=dist)) +
  geom_point(size = 4) +
  ggtitle("Cars data")
```



Apply scaling to entire data frame.

```
cars_norm <- cars %>% mutate(speed = scale(speed), dist=scale(dist))
cars_norm
```

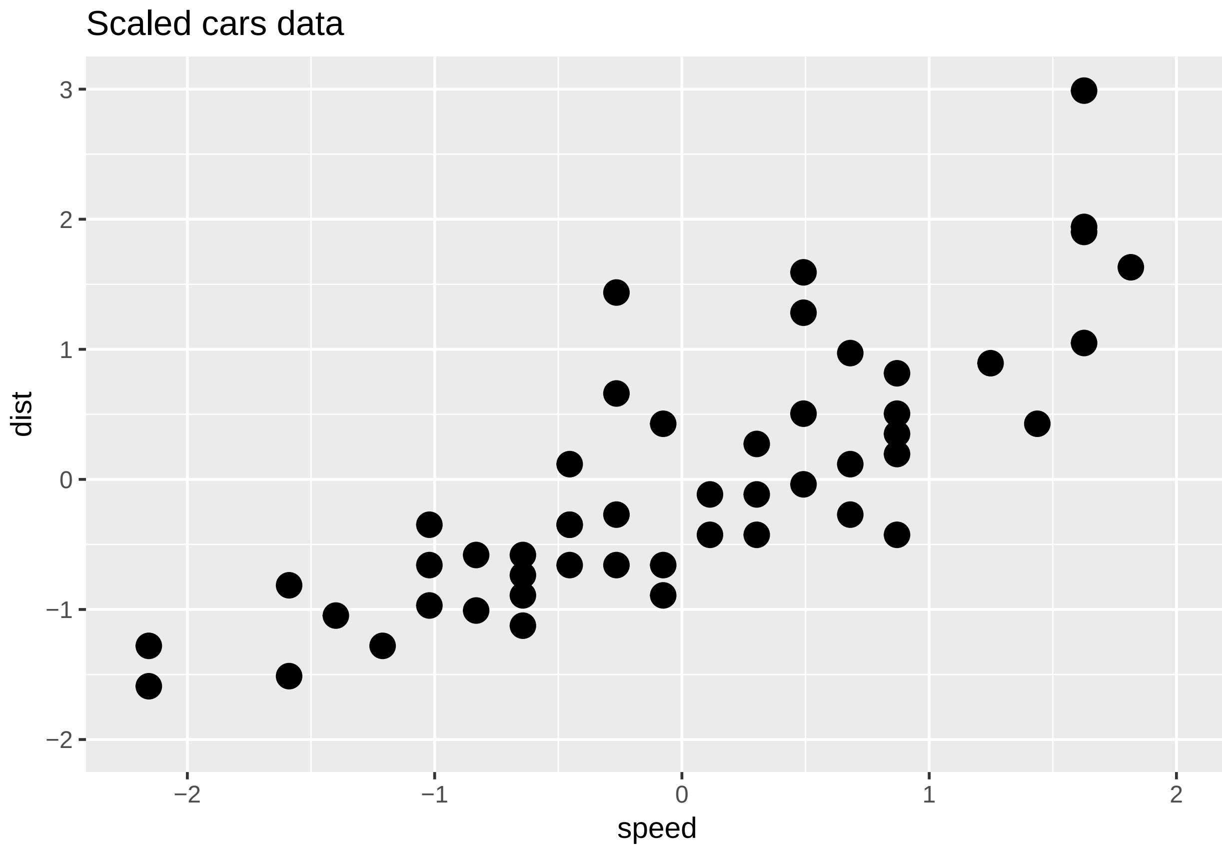
```
## # A tibble: 50 x 2
```

```
##    speed[,1] dist[,1]
##        <dbl>    <dbl>
##  1    -2.16    -1.59
##  2    -2.16    -1.28
##  3    -1.59    -1.51
##  4    -1.59    -0.814
##  5    -1.40    -1.05
##  6    -1.21    -1.28
##  7    -1.02    -0.969
##  8    -1.02    -0.659
##  9    -1.02    -0.348
## 10    -0.832   -1.01
## # ... with 40 more rows
```

```r
str(cars_norm)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    50 obs. of  2 variables:
##  $ speed: num [1:50, 1] -2.16 -2.16 -1.59 -1.59 -1.4 ...
##   ..- attr(*, "scaled:center")= num 15.4
##   ..- attr(*, "scaled:scale")= num 5.29
##  $ dist : num [1:50, 1] -1.59 -1.28 -1.513 -0.814 -1.047 ...
##   ..- attr(*, "scaled:center")= num 43
##   ..- attr(*, "scaled:scale")= num 25.8
```

```r
cars_norm %>% ggplot(aes(x=speed, y=dist)) +
  geom_point(size = 4) +
  ggtitle("Scaled cars data") +
  scale_x_continuous(limits = c(-2.2, 2)) +
  scale_y_continuous(limits = c(-2, 3))
```



Create training and test data.

Side note: This is not done using best practices, the scale() function should only be applied to the training data not the entire dataset. This is a common practice in many machine learning books. This should be corrected.

```
set.seed(12345)

idx <- sample(1:50, 40)

cars_train <- cars_norm[idx, ]
str(cars_train)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    40 obs. of  2 variables:
##  $ speed: num [1:40, 1] 0.681 0.87 1.816 0.87 -0.265 ...
##  $ dist : num [1:40, 1] 0.117 0.816 1.631 0.505 -0.271 ...
```

```
cars_test <- cars_norm[-idx, ]
str(cars_test)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  2 variables:
##  $ speed: num [1:10, 1] -1.589 -1.021 -1.021 -0.454 -0.454 ...
##  $ dist : num [1:10, 1] -1.513 -0.969 -0.348 -0.348 0.117 ...
```

```
cars_train <- cars_norm[idx, ]
str(cars_train)
```

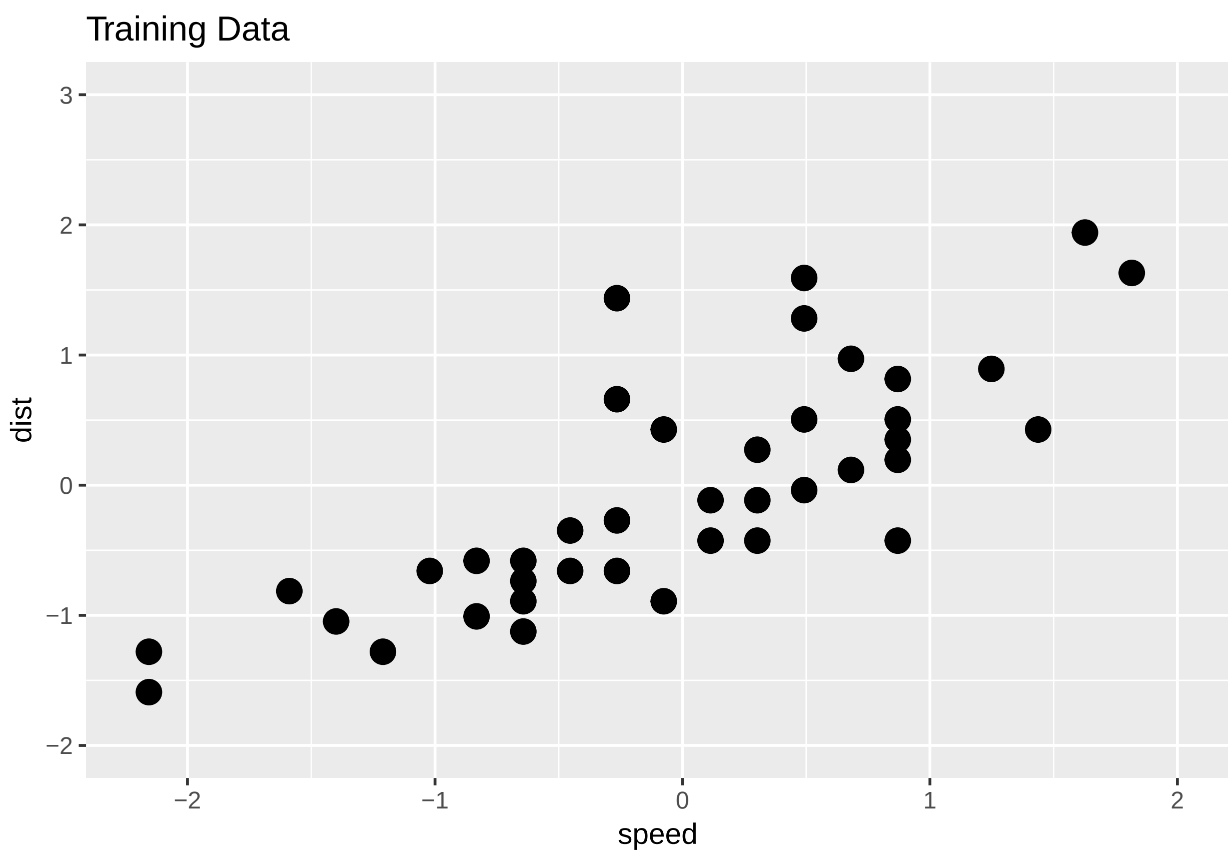
```
## Classes 'tbl_df', 'tbl' and 'data.frame':    40 obs. of  2 variables:
##  $ speed: num [1:40, 1] 0.681 0.87 1.816 0.87 -0.265 ...
```

```
##  $ dist : num [1:40, 1] 0.117 0.816 1.631 0.505 -0.271 ...
```

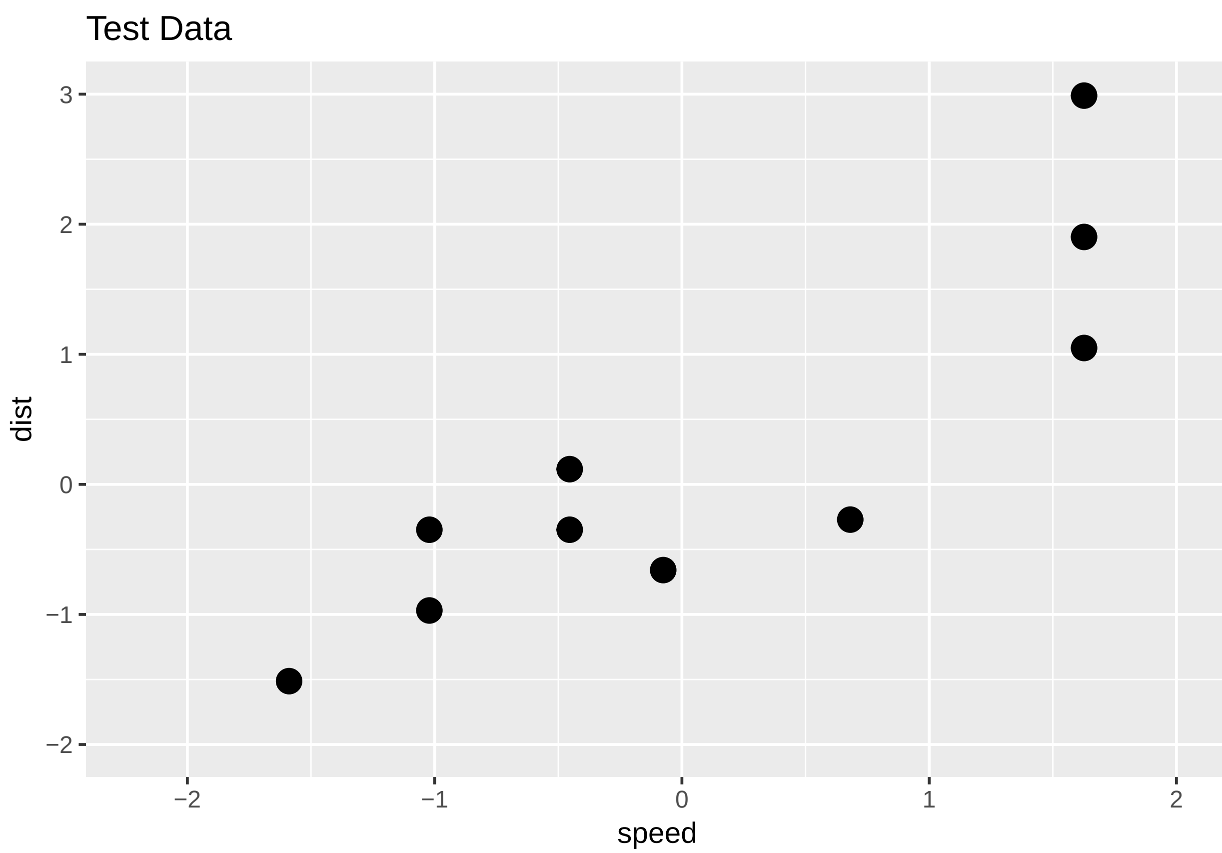
```
cars_test <- cars_norm[-idx, ]
str(cars_test)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  2 variables:
##  $ speed: num [1:10, 1] -1.589 -1.021 -1.021 -0.454 -0.454 ...
##  $ dist : num [1:10, 1] -1.513 -0.969 -0.348 -0.348 0.117 ...
```

```
cars_train %>% ggplot(aes(x=speed, y=dist)) +
  geom_point(size = 4) +
  ggtitle("Training Data") +
  scale_x_continuous(limits = c(-2.2, 2)) +
  scale_y_continuous(limits = c(-2, 3))
```




```
cars_test %>% ggplot(aes(x=speed, y=dist)) +
  geom_point(size = 4) +
  ggtitle("Test Data") +
  scale_x_continuous(limits = c(-2.2, 2)) +
  scale_y_continuous(limits = c(-2, 3))
```



Fit a simple linear regression. Train a linear regression model. Predict the Test Data. Compare predicted values with the holdout values.

```
cars_lm <- cars_train %>% lm(dist ~ speed, data = .)

summary(cars_lm)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.03373 -0.36619 -0.06137  0.23815  1.66240
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.03134    0.08919  -0.351    0.727
## speed        0.73450    0.09457   7.767 2.31e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5639 on 38 degrees of freedom
## Multiple R-squared:  0.6135, Adjusted R-squared:  0.6033
## F-statistic: 60.32 on 1 and 38 DF,  p-value: 2.315e-09
```

```r
predicted_lm_dist <- predict(cars_lm, cars_test)

# examine the correlation between predicted and actual values
cor(predicted_lm_dist, cars_test$dist)
```

```
##           [,1]
## [1,] 0.8696118
```

Fit a NN. Train a neural network model. Compare the R code. It is very similar.

```r
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 3.5.3
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

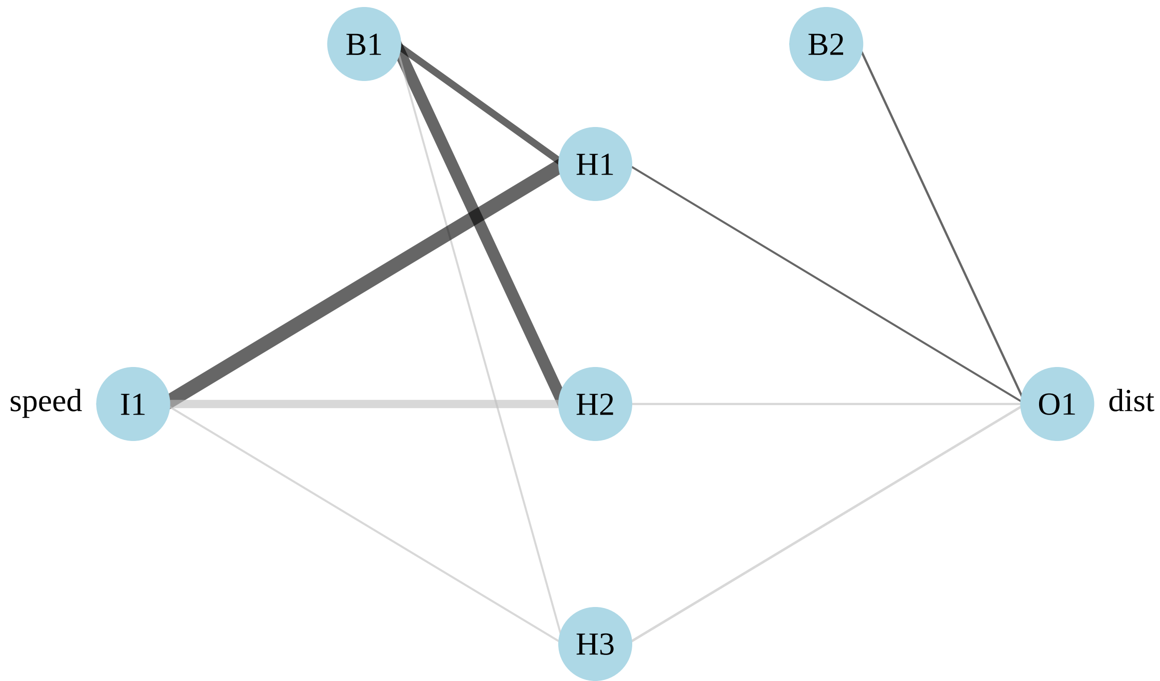
```r
set.seed(12345)

cars_model <- cars_train %>% neuralnet(formula = dist ~ speed,
        act.fct = "logistic", hidden = 3, linear.output=TRUE)

plot(cars_model)
```

Nice plot with the plotnet() function.

```r
library(NeuralNetTools)
```

```
## Warning: package 'NeuralNetTools' was built under R version 3.5.3
```

```r
par(mar = numeric(4), family = 'serif')
plotnet(cars_model, alpha = 0.6)
```



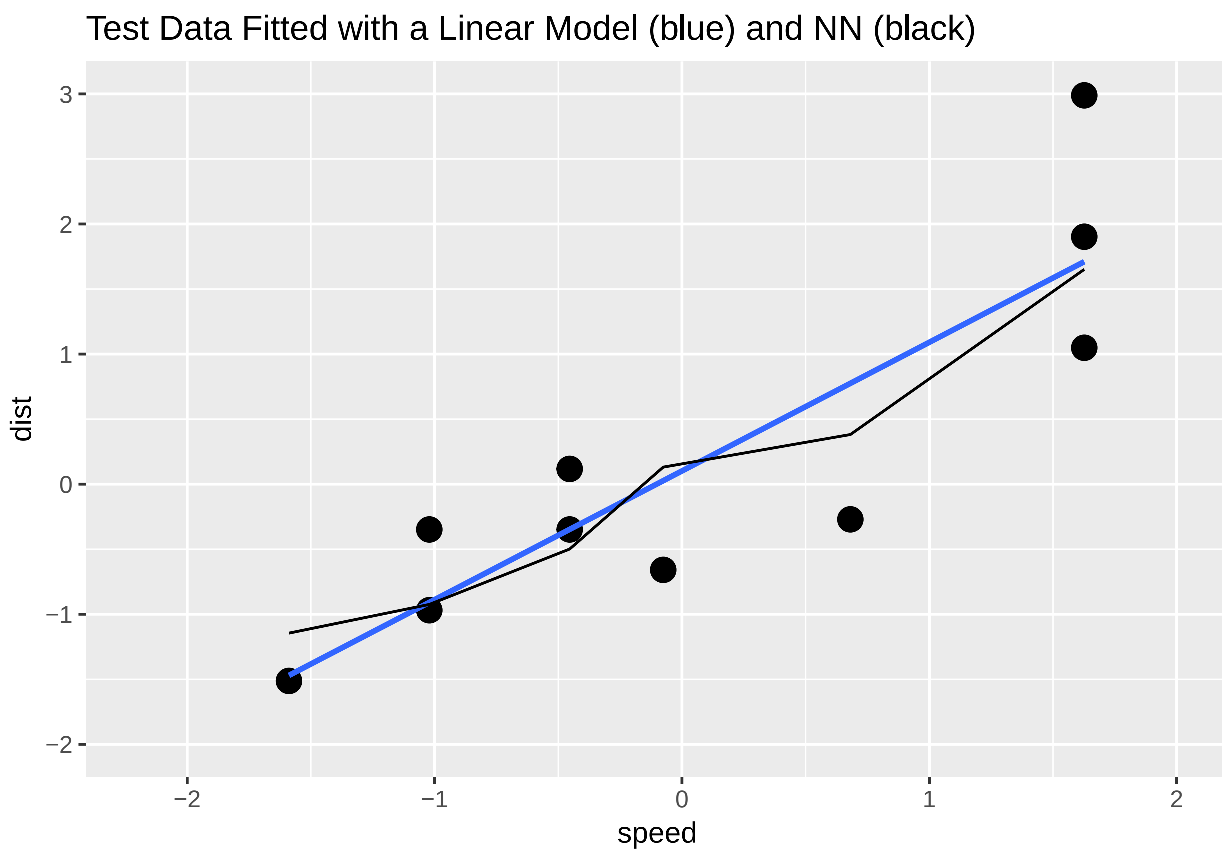
Predict the Test Data. Compare predicted values with the holdout values.

```
model_results <- compute(cars_model, cars_test[1])

predicted_dist <- model_results$net.result

# examine the correlation between predicted and actual values
cor(predicted_dist, cars_test$dist)
```

```
##           [,1]
## [1,] 0.8744087
```

Plot the fitted models.

```
ggplot(data=cars_test, aes(x=speed, y=dist)) +
  geom_point(size = 4) +
  geom_smooth(method='lm', formula=y~x, fill=NA) +
  geom_line(aes(y = predicted_dist)) +
  ggtitle("Test Data Fitted with a Linear Model (blue) and NN (black)") +
  scale_x_continuous(limits = c(-2.2, 2)) +
  scale_y_continuous(limits = c(-2, 3))
```



Example: Compare Simple Linear Regression to a Deep Learning, multilayer neural network.

a. Do you think this model will orverfit?

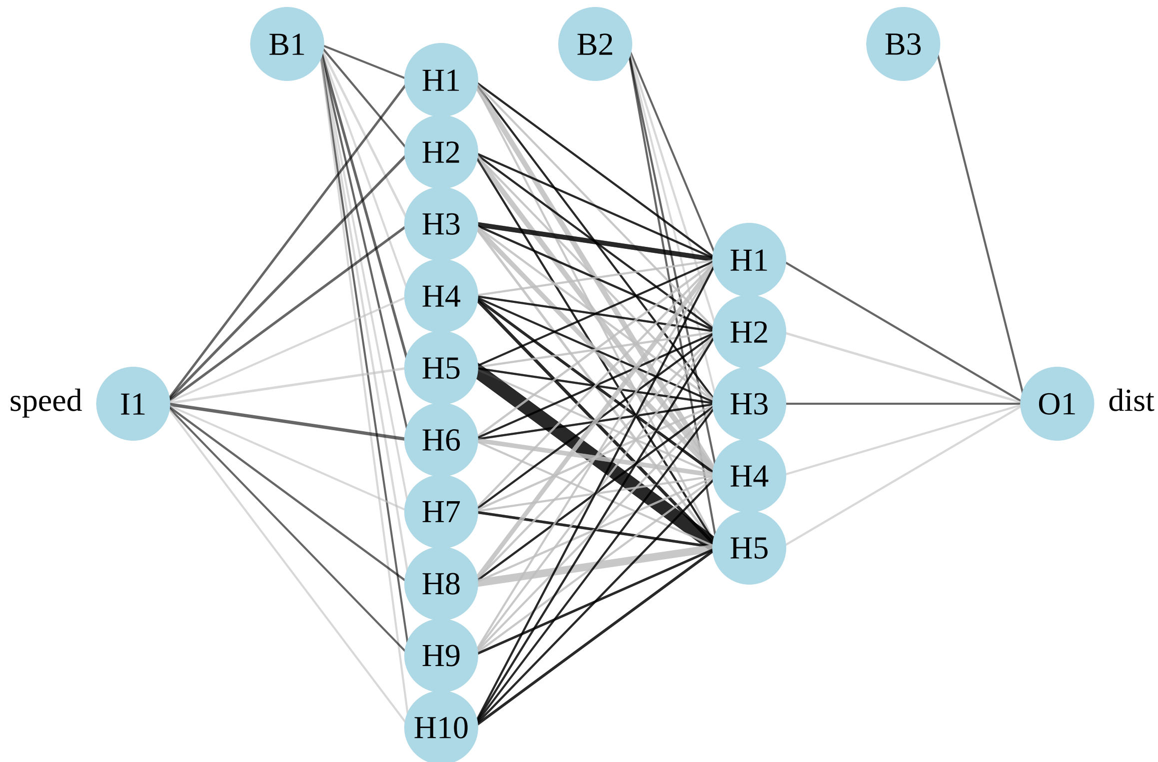
b. What does parsimonious mean?

c. Suggest a better measure for goodness-of-fit.

```
cars_model <- cars_train %>% neuralnet(formula = dist ~ speed,
        act.fct = "logistic", hidden = c(10,5), linear.output=TRUE)

plot(cars_model)
```

Nice plot with the plotnet() function.

```
par(mar = numeric(4), family = 'serif')
plotnet(cars_model, alpha = 0.6)
```



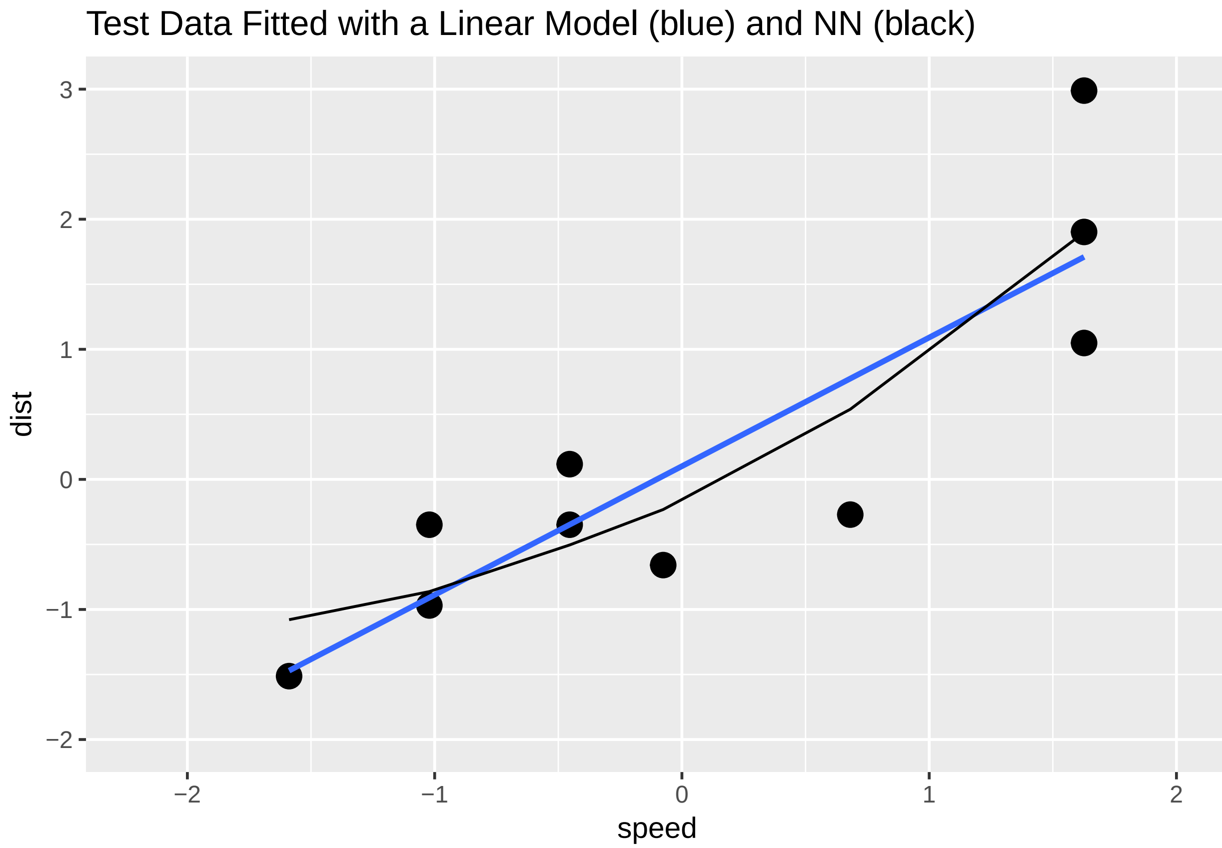
Predict the Test Data. Compare predicted values with the holdout values.

```r
model_results <- compute(cars_model, cars_test[1])

predicted_dist <- model_results$net.result

# examine the correlation between predicted and actual values
cor(predicted_dist, cars_test$dist)
```

```
##           [,1]
## [1,] 0.8890834
```

Plot the fitted models.

```r
ggplot(data=cars_test, aes(x=speed, y=dist)) +
  geom_point(size = 4) +
  geom_smooth(method='lm', formula=y~x, fill=NA) +
  geom_line(aes(y = predicted_dist)) +
  ggtitle("Test Data Fitted with a Linear Model (blue) and NN (black)") +
  scale_x_continuous(limits = c(-2.2, 2)) +
  scale_y_continuous(limits = c(-2, 3))
```



Run the concrete Example.

Step 2: Exploring and preparing the data —-

read in data and examine structure

```
concrete <- read.csv("http://www.sci.csueastbay.edu/~esuess/stat654/Poster/concrete.csv")
str(concrete)
```

```
## 'data.frame':    1030 obs. of  9 variables:
##  $ cement      : num  141 169 250 266 155 ...
##  $ slag        : num  212 42.2 0 114 183.4 ...
##  $ ash         : num  0 124.3 95.7 0 0 ...
##  $ water       : num  204 158 187 228 193 ...
##  $ superplastic: num  0 10.8 5.5 0 9.1 0 0 6.4 0 9 ...
##  $ coarseagg   : num  972 1081 957 932 1047 ...
##  $ fineagg     : num  748 796 861 670 697 ...
##  $ age         : int  28 14 28 28 28 90 7 56 28 28 ...
##  $ strength    : num  29.9 23.5 29.2 45.9 18.3 ...
```

custom normalization function

```
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
```

```r
concrete_norm <- as.data.frame(lapply(concrete, normalize))
summary(concrete_norm$strength)
```

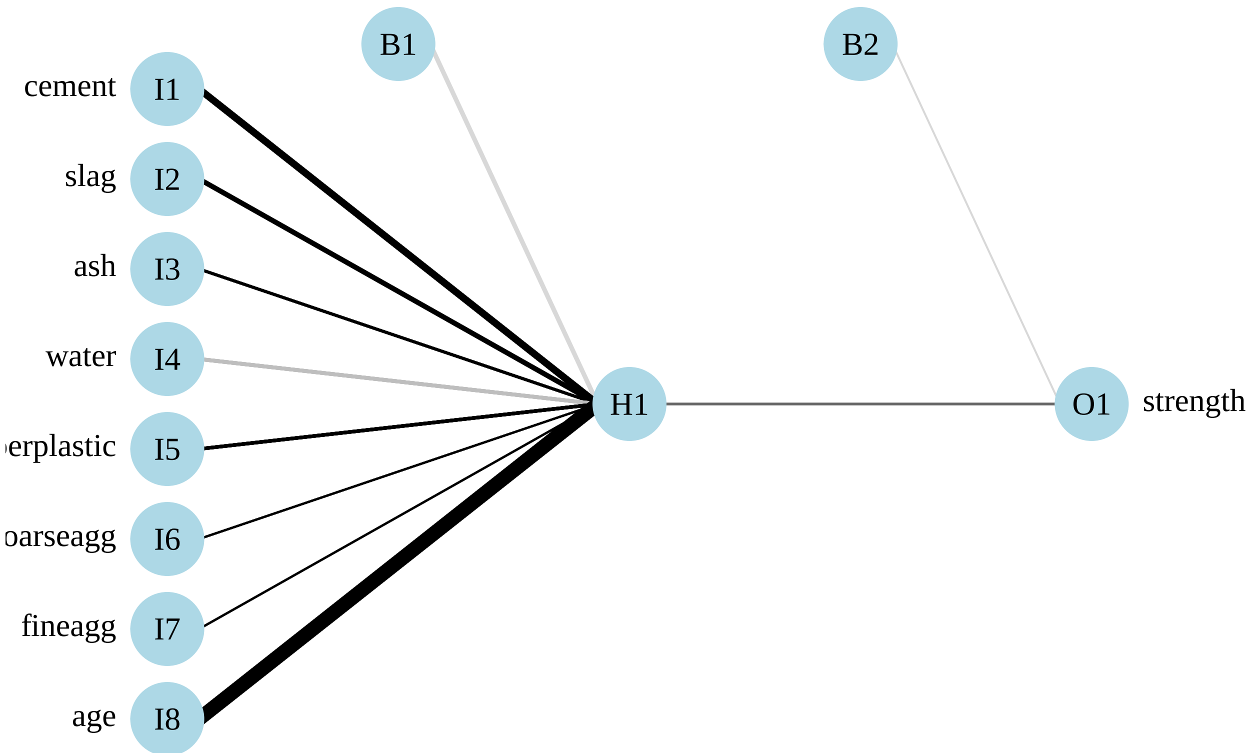
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.2664  0.4001  0.4172  0.5457  1.0000
```

```r
concrete_train <- concrete_norm[1:773, ]
concrete_test <- concrete_norm[774:1030, ]
```

Step 3: Training a model on the data —-

```r
library(neuralnet)

# simple ANN with only a single hidden neuron
set.seed(12345) # to guarantee repeatable results
concrete_model <- neuralnet(formula = strength ~ cement + slag +
                            ash + water + superplastic +
                            coarseagg + fineagg + age,
                            data = concrete_train)

# visualize the network topology
plot(concrete_model)
```

```r
# alternative plot
library(NeuralNetTools)

# plotnet
par(mar = numeric(4), family = 'serif')
plotnet(concrete_model, alpha = 0.6)
```



Step 4: Evaluating model performance —-

```
# obtain model results
model_results <- compute(concrete_model, concrete_test[1:8])
# obtain predicted strength values
predicted_strength <- model_results$net.result
# examine the correlation between predicted and actual values
cor(predicted_strength, concrete_test$strength)   # higher than stated in book 0.7170368646
```

```
##           [,1]
## [1,] 0.8064656
```

```
# produce actual predictions by

head(predicted_strength)
```

```
##          [,1]
## 774 0.3258992
## 775 0.4677425
## 776 0.2370268
## 777 0.6718811
## 778 0.4663429
## 779 0.4685272
```

```
concrete_train_original_strength <- concrete[1:773,"strength"]

strength_min <- min(concrete_train_original_strength)
strength_max <- max(concrete_train_original_strength)
```

```r
head(concrete_train_original_strength)
```

```
## [1] 29.89 23.51 29.22 45.85 18.29 21.86
```

```r
# custom normalization function
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}

strength_pred <- unnormalize(predicted_strength, strength_min, strength_max)
#strength_pred
```

Step 5: Improving model performance —-

```r
# a more complex neural network topology with 5 hidden neurons
set.seed(12345) # to guarantee repeatable results
concrete_model2 <- neuralnet(strength ~ cement + slag +
                             ash + water + superplastic +
                             coarseagg + fineagg + age,
                             data = concrete_train, hidden = 5, act.fct = "logistic")

# plot the network
plot(concrete_model2)

# plotnet
par(mar = numeric(4), family = 'serif')
plotnet(concrete_model2, alpha = 0.6)

# evaluate the results as we did before
model_results2 <- compute(concrete_model2, concrete_test[1:8])
predicted_strength2 <- model_results2$net.result
cor(predicted_strength2, concrete_test$strength)  # higher than stated in book 0.801444583
```

```
##           [,1]
## [1,] 0.9244533
```

```r
# try different activation function
# a more complex neural network topology with 5 hidden neurons
set.seed(12345) # to guarantee repeatable results
concrete_model2 <- neuralnet(strength ~ cement + slag +
                             ash + water + superplastic +
                             coarseagg + fineagg + age,
                             data = concrete_train, hidden = 5, act.fct = "tanh")

# evaluate the results as we did before
model_results2 <- compute(concrete_model2, concrete_test[1:8])
predicted_strength2 <- model_results2$net.result
cor(predicted_strength2, concrete_test$strength)
```

```
##           [,1]
## [1,] 0.5741729
```

using h2o deeplearning

```r
library(h2o)
```

```
## Warning: package 'h2o' was built under R version 3.5.3

##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------

##
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```r
h2o.init(nthreads=8, max_mem_size="2G")
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 54 minutes
##     H2O cluster timezone:       America/Los_Angeles
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.22.1.1
##     H2O cluster version age:    2 months and 19 days
##     H2O cluster name:           H2O_started_from_R_jiayi_zqn968
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.61 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:         Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.5.1 (2018-07-02)
```

```r
h2o.removeAll() ## clean slate - just in case the cluster was already running
```

```
## [1] 0
```

```r
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 54 minutes
##     H2O cluster timezone:       America/Los_Angeles
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.22.1.1
##     H2O cluster version age:    2 months and 19 days
##     H2O cluster name:           H2O_started_from_R_jiayi_zqn968
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.61 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:         Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.5.1 (2018-07-02)
```

```r
concrete.hex <- h2o.importFile("http://www.sci.csueastbay.edu/~esuess/stat654/Poster/concrete.csv")
```

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

```r
summary(concrete.hex)
```

```
## Warning in summary.H2OFrame(concrete.hex): Approximated quantiles
## computed! If you are interested in exact quantiles, please pass the
## `exact_quantiles=TRUE` parameter.
```

```
##  cement          slag             ash              water
##  Min.   :102.0   Min.   :  0.00   Min.   :  0.00   Min.   :121.8
##  1st Qu.:192.1   1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:164.9
##  Median :272.6   Median : 21.92   Median :  0.00   Median :184.9
##  Mean   :281.2   Mean   : 73.90   Mean   : 54.19   Mean   :181.6
##  3rd Qu.:349.9   3rd Qu.:142.68   3rd Qu.:118.26   3rd Qu.:191.9
##  Max.   :540.0   Max.   :359.40   Max.   :200.10   Max.   :247.0
##  superplastic     coarseagg        fineagg          age
##  Min.   : 0.000   Min.   : 801.0   Min.   :594.0   Min.   :  1.00
##  1st Qu.: 0.000   1st Qu.: 931.7   1st Qu.:730.8   1st Qu.:  7.00
##  Median : 6.376   Median : 967.8   Median :779.1   Median : 28.00
##  Mean   : 6.205   Mean   : 972.9   Mean   :773.6   Mean   : 45.66
##  3rd Qu.:10.175   3rd Qu.:1029.1   3rd Qu.:824.0   3rd Qu.: 56.00
##  Max.   :32.200   Max.   :1145.0   Max.   :992.6   Max.   :365.00
##  strength
```

```
##  Min.   : 2.33
##  1st Qu.:23.68
##  Median :34.40
##  Mean   :35.82
##  3rd Qu.:46.10
##  Max.   :82.60
```

```
splits <- h2o.splitFrame(concrete.hex, 0.75, seed=1234)

dl <- h2o.deeplearning(x=1:8,y="strength",training_frame=splits[[1]],activation = "Tanh",
                       hidden = c(200,200), distribution = "gaussian")
```

```
##
  |
  |                                                                 |   0%
  |
  |======                                                           |  10%
  |
  |================================                                 |  50%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%
```

```
dl.predict <- h2o.predict(dl, splits[[2]])
```

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

```
cor(as.vector(dl.predict), as.vector(splits[[2]]$strength))
```

```
## [1] 0.9060864
```

```
dl@parameters
```

```
## $model_id
## [1] "DeepLearning_model_R_1553049245964_3"
##
## $training_frame
## [1] "RTMP_sid_ad96_2"
##
## $activation
## [1] "Tanh"
##
## $seed
## [1] -7.975319e+18
##
## $distribution
## [1] "gaussian"
##
## $x
## [1] "cement"       "slag"         "ash"          "water"
## [5] "superplastic" "coarseagg"    "fineagg"      "age"
##
```

```
## $y
## [1] "strength"
```

```r
h2o.performance(dl)
```

```
## H2ORegressionMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  42.59492
## RMSE:  6.526479
## MAE:  5.150411
## RMSLE:  0.2309562
## Mean Residual Deviance :  42.59492
```

```r
#h2o.shutdown()
```

Run the iris Example.

```r
library("keras")
```

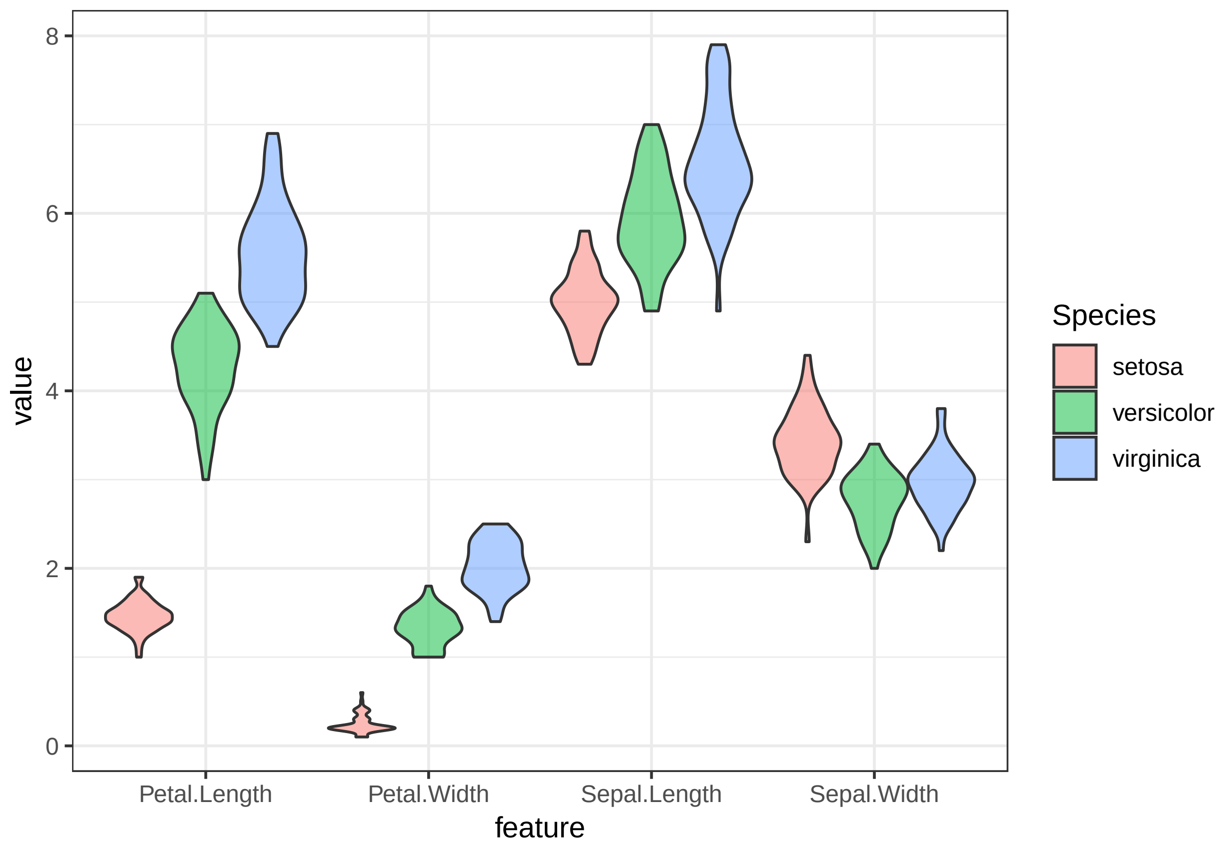
```
## Warning: package 'keras' was built under R version 3.5.3
```

```
##
## Attaching package: 'keras'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     normalize
```

```r
suppressMessages(library("tidyverse"))
```

```r
iris %>% as_tibble %>% gather(feature, value, -Species) %>%
  ggplot(aes(x = feature, y = value, fill = Species)) +
  geom_violin(alpha = 0.5, scale = "width") +
  theme_bw()
```



Prepare data

We start with slightly wrangling the iris data set by renaming and scaling the features and converting character labels to numeric:

```
set.seed(265509)
nn_dat <- iris %>% as_tibble %>%
  mutate(sepal_length = scale(Sepal.Length),
         sepal_width  = scale(Sepal.Width),
         petal_length = scale(Petal.Length),
         petal_width  = scale(Petal.Width),
         class_label  = as.numeric(Species) - 1) %>%
    select(sepal_length, sepal_width, petal_length, petal_width, class_label)

nn_dat %>% head(3)
```

```
## # A tibble: 3 x 5
##   sepal_length[,1] sepal_width[,1] petal_length[,1] petal_width[,1]
##              <dbl>           <dbl>            <dbl>           <dbl>
## 1           -0.898           1.02             -1.34           -1.31
## 2           -1.14           -0.132            -1.34           -1.31
## 3           -1.38            0.327            -1.39           -1.31
## # ... with 1 more variable: class_label <dbl>
```

Then, we create indices for splitting the iris data into a training and a test data set. We set aside 20% of the data for testing:

```
test_fraction   <- 0.20
n_total_samples <- nrow(nn_dat)
n_train_samples <- ceiling((1 - test_fraction) * n_total_samples)
train_indices   <- sample(n_total_samples, n_train_samples)
n_test_samples  <- n_total_samples - n_train_samples
test_indices    <- setdiff(seq(1, n_train_samples), train_indices)
```

Based on the indices, we can now create training and test data

```
x_train <- nn_dat %>% select(-class_label) %>% as.matrix %>% .[train_indices,]
y_train <- nn_dat %>% pull(class_label) %>% .[train_indices] %>% to_categorical(3)
x_test  <- nn_dat %>% select(-class_label) %>% as.matrix %>% .[test_indices,]
y_test  <- nn_dat %>% pull(class_label) %>% .[test_indices] %>% to_categorical(3)
```

Set Architecture

With the data in place, we now set the architecture of our artifical neural network:

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 4, activation = 'relu', input_shape = 4) %>%
  layer_dense(units = 3, activation = 'softmax')
model %>% summary
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense_1 (Dense)                  (None, 4)                     20
## ___________________________________________________________________________
## dense_2 (Dense)                  (None, 3)                     15
## ===========================================================================
## Total params: 35
## Trainable params: 35
## Non-trainable params: 0
## ___________________________________________________________________________
```

Next, the architecture set in the model needs to be compiled:

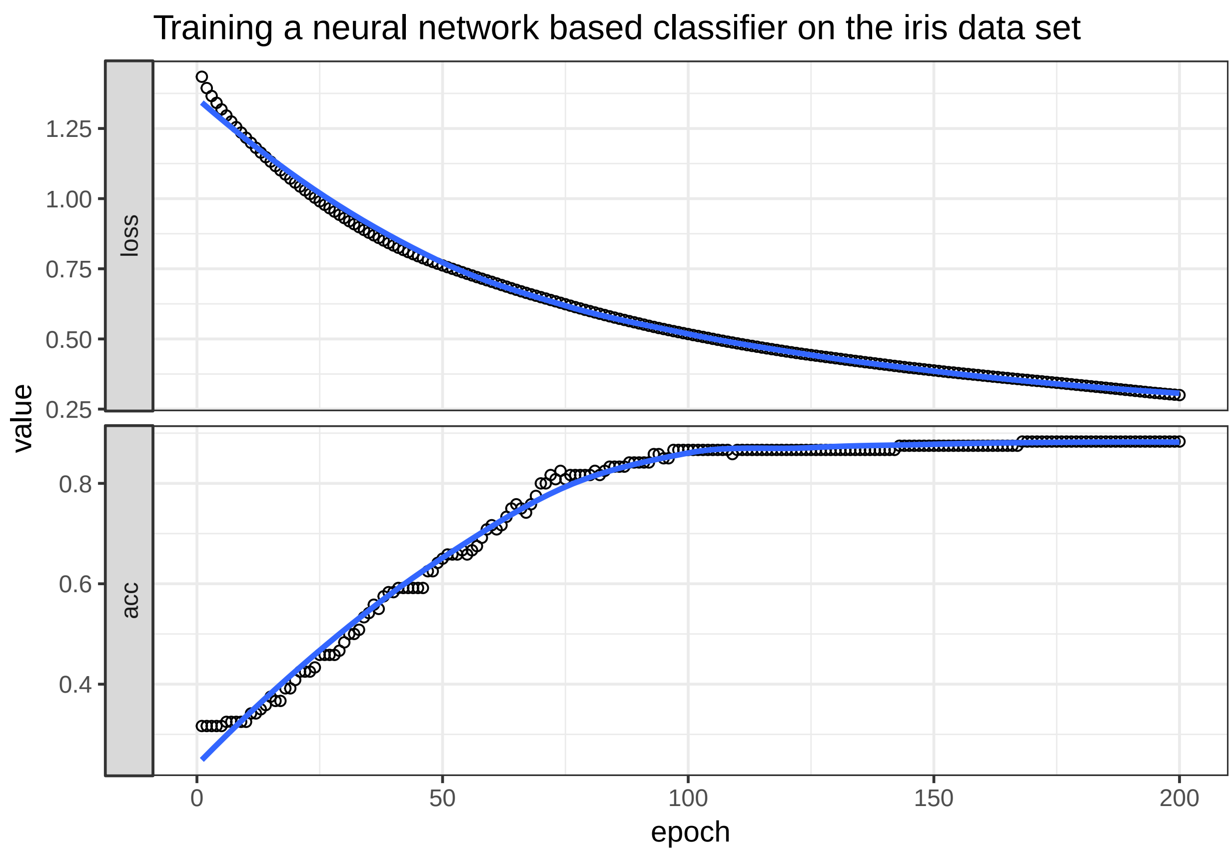
```
model %>% compile(
  loss      = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics   = c('accuracy')
)
```

Train the Artificial Neural Network

Lastly we fit the model and save the training progres in the `history` object:

```
history <- model %>% fit(
  x = x_train, y = y_train,
  epochs = 200,
  batch_size = 20,
  validation_split = 0
)
plot(history) +
```

```
  ggtitle("Training a neural network based classifier on the iris data set") +
  theme_bw()
```



Evaluate Network Performance

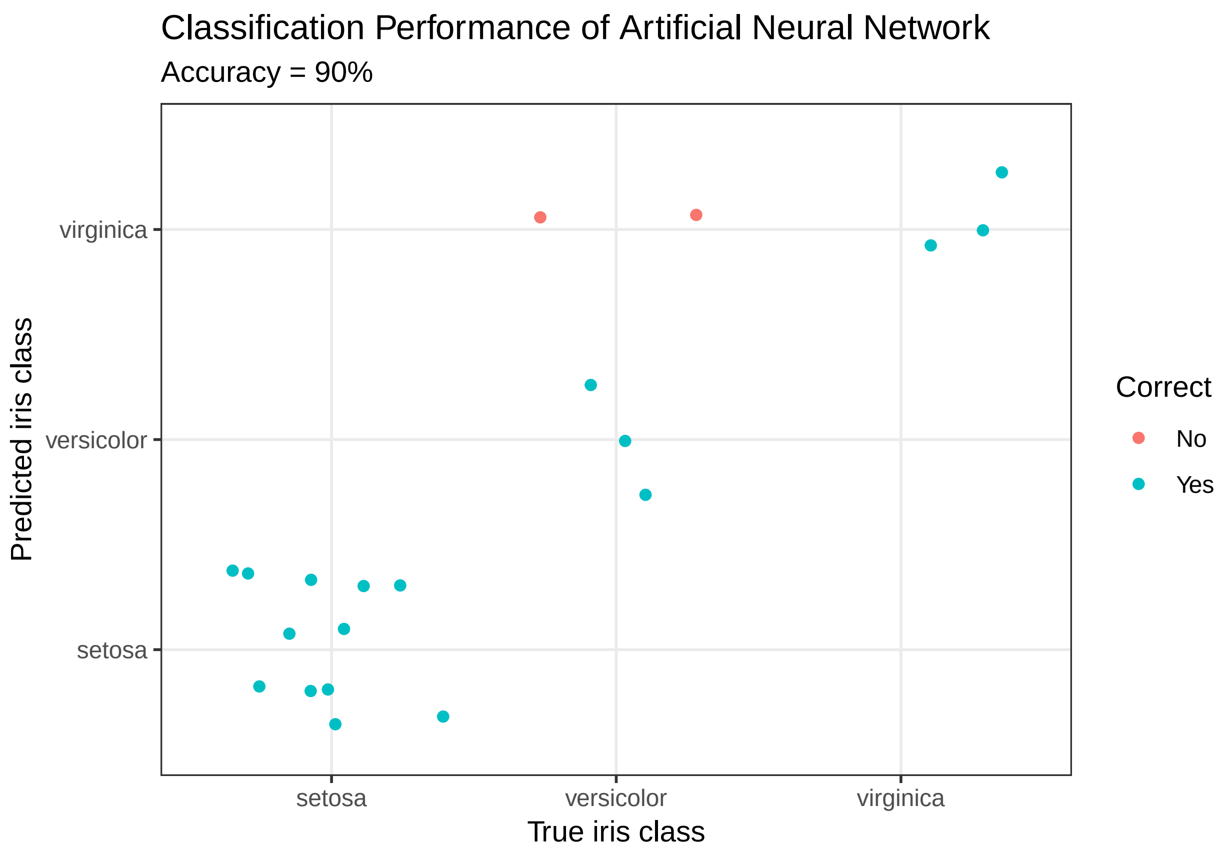
The final performance can be obtained like so:

```
perf <- model %>% evaluate(x_test, y_test)
print(perf)
```

```
## $loss
## [1] 0.2228689
##
## $acc
## [1] 0.9
```

```
classes <- iris %>% as_tibble %>% pull(Species) %>% unique
y_pred  <- model %>% predict_classes(x_test)
y_true  <- nn_dat %>% pull(class_label) %>% .[test_indices]

tibble(y_true = classes[y_true + 1], y_pred = classes[y_pred + 1],
       Correct = ifelse(y_true == y_pred, "Yes", "No") %>% factor) %>%
  ggplot(aes(x = y_true, y = y_pred, colour = Correct)) +
  geom_jitter() +
  theme_bw() +
  ggtitle(label = "Classification Performance of Artificial Neural Network",
```

```r
         subtitle = str_c("Accuracy = ",round(perf$acc,3)*100,"%")) +
  xlab(label = "True iris class") +
  ylab(label = "Predicted iris class")
```




```r
library(gmodels)
```

```
## Warning: package 'gmodels' was built under R version 3.5.3
```

```r
CrossTable(y_pred, y_true,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
           dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  20
##
##
##              | actual
##    predicted |         0 |         1 |         2 | Row Total |
## -------------|-----------|-----------|-----------|-----------|
```

```
##            0 |        12 |         0 |         0 |        12 |
##              |     1.000 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            1 |         0 |         3 |         0 |         3 |
##              |     0.000 |     0.600 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            2 |         0 |         2 |         3 |         5 |
##              |     0.000 |     0.400 |     1.000 |           |
## -------------|-----------|-----------|-----------|-----------|
## Column Total |        12 |         5 |         3 |        20 |
##              |     0.600 |     0.250 |     0.150 |           |
## -------------|-----------|-----------|-----------|-----------|
##
##
```

Run the code in Chapter 2. A first look at a neural network

2.1 load train and test data

```
library(keras)
mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y
```

data prepare

```
str(train_images)
```

```
##  int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
str(train_labels)
```

```
##  int [1:60000(1d)] 5 0 4 1 9 2 1 3 1 4 ...
```

```
str(test_images)
```

```
##  int [1:10000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
str(test_labels)
```

```
##  int [1:10000(1d)] 7 2 1 0 4 1 4 9 5 9 ...
```

```
train_images <- array_reshape(train_images, c(60000, 28 *28))
train_images <- train_images/255

test_images <- array_reshape(test_images, c(10000, 28*28))
test_images <- test_images/255
```

```
train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)
```

model set

```
network <- keras_model_sequential()%>%
  layer_dense(units=512, activation = "relu", input_shape = c(28*28)) %>%
  layer_dense(units = 10, activation = "softmax")
```

```
network %>% compile(
  optimizer= "rmsprop",
  loss = "categorical_crossentropy",
  metrics= c("accuracy")
)
```

fit model

```
network %>% fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
metrics <- network %>% evaluate(test_images, test_labels)
metrics
```

```
## $loss
## [1] 0.06781651
##
## $acc
## [1] 0.9798
```

```
network %>% predict_classes(test_images[1:10,])
```

```
##  [1] 7 2 1 0 4 1 4 9 5 9
```

2.2 data preparation

```
x <- matrix(rep(0,3*5), nrow = 3, ncol = 5)
x
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
```

```
x<- array(rep(0,2*3*2), dim = c(2,3,2))
str(x)
```

```
##  num [1:2, 1:3, 1:2] 0 0 0 0 0 0 0 0 0 0 ...
```

```
dim(x)
```

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

```
length(dim(train_images))
```

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

```
dim(train_images)
```

```
## [1] 60000   784
```

```r
typeof(train_images)
```

```
## [1] "double"
```

```r
#digit <- train_images[5,,]
#plot(as.raster(digit, max = 255))
#my_slice <- train_images[10:99,,]
#dim(my_slice)
#my_slice <- train_images[10:99,1:28,1:28]
#dim(my_slice)
#my_slice <- train_images[,15:28,15:28]
```

```r
#batch <- train_images[1:128,,]
#batch <- train_images[129:256,,]
```

2.3 gears of neural networks

```r
layer_dense(units=512, activation = "relu")
```

```
## <keras.layers.core.Dense>
```

```r
#output= relu(dot(w, input)+ b)
```

```r
naive_relu <- function(x){
  for (i in nrow(x))
    for (j in ncol(x))
      x[i,j] <- max(x[i,j],0)
    x

}
```

```r
naive_relu <- function(x){
  for (i in nrow(x))
    for (j in ncol(x))
      x[i,j] = x[i,j]+y[i,j]
    x

}
```

```r
x <-array(round(runif(1000,0,9)), dim = c(64,3,32,10))
y <- array(5, dim = c(32,10))
z <- sweep(x, c(3,4), y, pmax)
```

```r
#z <- x+y
#z <- pmax(z,0)
```

```r
#sweep(x, 2, y, '+')
```

```r
naive_vector_dot <- function(x,y){
  z<- 0
  for (i in 1:length(x))
    z <- z+x[[i]]*y[[i]]
  z
}
```

```r
naive_matrix_vector_dot <- function(x,y){
  z<- rep(0, nrow(x))
  for (i in 1:nrow(x))
    for (j in 1:ncol(x))
    z <- z[[i]]+x[[i,j]]*y[[j]]
  z
}
```

```r
naive_matrix_vector_dot <- function(x, y) {
  z <- rep(0, nrow(x))
  for (i in 1:nrow(x))
    z[[i]] <- naive_vector_dot(x[i,], y)
  z
}
```

```r
naive_matrix_dot <- function(x, y) {
  z <- matrix(0, nrow = nrow(x), ncol = ncol(y))
  for (i in 1:nrow(x))
    for (j in 1:ncol(y)) {
      row_x <- x[i,]
      column_y <- y[,j]
      z[i, j] <- naive_vector_dot(row_x, column_y)
    }
  z
  }
```

```r
train_images <- array_reshape(train_images, c(60000, 28 * 28))
```

```r
x <- matrix(c(0, 1,
              2, 3,
              4, 5),
            nrow = 3, ncol = 2, byrow = TRUE)
x <- array_reshape(x, dim = c(6, 1))
x <- array_reshape(x, dim = c(2, 3))
x <- matrix(0, nrow = 300, ncol = 20)
dim(x)
```

```
## [1] 300  20
```

```r
x <- t(x)
```

2.5 Looking back at our first example

```r
network %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))
```

```r
compile(
  network,
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))
```

```r
network %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)
```