Assignment 2

Exercise 1

Arthur Junges Schmidt

08 July 2020

Exercise 1a

```
Input1 <- cbind(5, 0.5, 2)
Desired_Output <- 1
learn_rate <- 5
Hidden_Nodes <- 2
Number_Iterations <- 2
Weights_Input <- matrix(0.1, nrow = ncol(Input1), ncol = Hidden_Nodes)
Weights_Output <- matrix(0.5, Hidden_Nodes, 1)



train <- function(x, y, hidden, learn_rate, iterations, Weights1, Weights2) {
  d <- ncol(x)
  w1 <- matrix(Weights1, d, hidden)
  w2 <- matrix(Weights2, nrow = hidden , 1)
  for (i in 1:iterations) {
    ff <- feed_forward(x, w1, w2)
    bp <- feed_backward(x, y,
                        y_hat = ff$output,
                        w1, w2,
                        h = ff$h,
                        learn_rate = learn_rate)
    w1 <- bp$w1; w2 <- bp$w2
  }
  list(output = ff$output, w1 = w1, w2 = w2)

}


feed_forward <- function(x = Input1, w1, w2){
  z1 <- cbind(x) %*% w1
  h <- sigmoid(z1)
  z2 <- cbind(h) %*% w2
  list(output = sigmoid(z2), h = h)
}

sigmoid <- function(x) {
  1 / (1 + exp(-x))
}

feed_backward <- function(x, y = Desired_Output, y_hat, w1, w2, h, learn_rate) {

  dw2 <- (y_hat - y) * y_hat * (1 - y_hat) * as.vector(h)

  dh <- (as.double(y_hat) - y) * as.double(y_hat) * (1 - as.double(y_hat)) %*% w2[1]
```

```
  dw1 <- t(x) %*% (h *  (1 - h) * c(dh))



  w1 <- w1 - learn_rate * dw1
  w2 <- w2 - learn_rate * dw2

  list(w1 = w1, w2 = w2)
}

nnet <- train(x = Input1, y = Desired_Output,
              hidden = Hidden_Nodes,
              learn_rate = learn_rate,
              iterations = Number_Iterations,
              Weights1 = Weights_Input,
              Weights2 = Weights_Output)
```

Exercise 1b

```
# Clear all ---------------------------------------------------------------
rm(list=ls())
gc()
cat("\014")

# Inputs ------------------------------------------------------------------
library(profvis)
#Input1 <- cbind(5, 0.5, 2)
#Desired_Output <- 1
learn_rate <- 5
Hidden_Nodes <- 4
#Number_Iterations <- 2
#Weights_Input <- matrix(0.1, nrow = ncol(Data_set_Y), ncol = Hidden_Nodes)
#Weights_Output <- matrix(0.5, Hidden_Nodes, 1)

# Sigmoid Function --------------------------------------------------------


sigmoid <- function(x) {
  1 / (1 + exp(-x))
}



# Generate sample data ----------------------------------------------------
a1 <- cbind(c(3, 5), c(2, 7), c(3, 8));
a2 <- cbind(c(3, 5), c(-2, -7), c(3, 8));

# Initialize Y variable with 2 rows and 1000 columns
Data_set_Y <- data.frame(replicate(1000, numeric(2))); # Y has 1000 columns and 2 values

# Initialize X variable with 3 rows and 1000 columns
Input_Values_X <- matrix(data = 0, nrow = 3, ncol = 1000);

# Assign the variable with the provided equation
for (n in 1:length(Data_set_Y)) {
  Input_Values_X[,n] <- rnorm(3);
  Data_set_Y[n] <- sigmoid(a1 %*% Input_Values_X[,n]) +
    ((a2 %*% Input_Values_X[,n])^2) + 0.30 * rnorm(2);
}

### ok!
# Dividing Sample data -----------------------------------------------------

#Data will be divided in 70% training data and 30% test data
```

```r
## Sample size
Sample_size <- floor(0.75 * NCOL(Data_set_Y));

## Set the seed to make the partition reproducible
set.seed(91374)

Training_Columns <- sample(seq_len(ncol(Data_set_Y)), Sample_size);
# Choose randomly which collumns will compose the training set

Output_Train_set <- Data_set_Y[, Training_Columns];
Output_Test_set <- Data_set_Y[, -Training_Columns];
Input_Train_set <- Input_Values_X[, Training_Columns];
Input_Test_set <- Input_Values_X[, -Training_Columns];


# Training ----------------------------------------------------------------

# The weights now are N(0,1)

train <- function(Input, Output, hidden, learn_rate, iterations) {
  Number_Inputs <- nrow(Input) # Now inputs are organized by columnns, so the size is 'nrow'
  Weight_Input_Hidden <- matrix(rnorm(Number_Inputs * hidden), hidden, Number_Inputs)

  Weight_Hidden_Output <- matrix(rnorm(hidden * 2), nrow = 2, ncol = hidden)
  # Size of output is 2

  ## Initialize Train Error Matrix --

  Error_Test_Matrix <- matrix(0, nrow = ncol(Input), ncol = 2);

  ## For loops --

  for (j in 1:iterations) {


    for (i in 1:ncol(Input)) {
    ff <- feed_forward(Input, w1 = Weight_Input_Hidden, w2 = Weight_Hidden_Output,
                       Sample_Number = i)

    bp <- feed_backward(Input, Output,
                        y_hat = ff$Output_Activated,
                        Weight_Input_Hidden, Weight_Hidden_Output,
                        Hidden_Activated = ff$Hidden_Activated,
                        learn_rate = learn_rate,
                        Sample_Number = i)
    Weight_Input_Hidden <- bp$w1;
    Weight_Hidden_Output <- bp$w2

    Error_Test_Matrix[i,] <- c(i, bp$error);

    }
    # print(Weight_Input_Hidden)
    # print(Weight_Hidden_Output)
    # print(j)
    # print(Error_Test_Matrix)
```

```r
  }
  list(output = ff$output, w1 = Weight_Input_Hidden,
       w2 = Weight_Hidden_Output,
       Error_Test_Matrix = Error_Test_Matrix)

}


feed_forward <- function(x, w1, w2, Sample_Number){

  Hidden_Activated <- sigmoid(w1 %*% x[,Sample_Number]);
  Output_Activated <- sigmoid(w2 %*% Hidden_Activated); ### OK!
  # z1 <- cbind(x) %*% w1
  # h <- sigmoid(z1)
  # z2 <- cbind(h) %*% w2
  # list(output = sigmoid(z2), h = h)

  result <- list(Hidden_Activated = Hidden_Activated, Output_Activated = Output_Activated);
  return(result)
}


feed_backward <- function(Input, Output, y_hat, w1, w2, Hidden_Activated,
                          learn_rate,
                          Sample_Number) {

  Error <- (sum((y_hat - Output[, Sample_Number])^2))*0.5

  dw2 <- ((y_hat - Output[,Sample_Number]) * y_hat * (1 - y_hat)) %*% t(Hidden_Activated)
  ## Hidden_activated is [4x1] so it needs to be transposed to obtain a result dw2[2x4]
  ## > Dot Multiplication!

  dw1 <- ((t(w2) %*% (y_hat - Output[, Sample_Number])) *
            (Hidden_Activated * (1 - Hidden_Activated))) %*%
    t(Input[, Sample_Number]);


  w1 <- w1 + learn_rate * dw1;
  w2 <- w2 + learn_rate * dw2;

  list(w1 = w1, w2 = w2, error = Error)
}
profvis({
Results_0.5 <- train(Input = Input_Train_set, Output = Output_Train_set, hidden = Hidden_Nodes,
                 learn_rate = 0.5,
                 iterations = 1000)
})
Results_5 <- train(Input = Input_Train_set, Output = Output_Train_set, hidden = Hidden_Nodes,
                   learn_rate = 5,
                   iterations = 1000)
```

Exercise 2

Arthur Junges Schmidt

08 July 2020

First step is the data processing of the CSV file. To separate the data sets into training, validation and forecasting, these calculations were made:

- The last 3 days of data correspond to the last $(24 \times 3) = 72$ rows of the original data set;
- The validation data has $(744 - 72) \times 20\% \approx 135$ rows;
- The train data is what's left: $(744 - 72 - 135) = 537$.

```
# Read data from CSV
# ------------------------------------------------------

Crude_Data <- read_csv("Problemset 2 data for problem 2.csv",
    skip = 1)
```

```
## Warning: Duplicated column names deduplicated: 'volume(veh/h)' => 'volume(veh/
## h)_1' [2], 'volume(veh/h)' => 'volume(veh/h)_2' [3], 'speed(km/h)' => 'speed(km/
## h)_1' [5], 'speed(km/h)' => 'speed(km/h)_2' [6]
```

```
for (i in 1:length(Crude_Data)) {
    if (i <= 3) {
        colnames(Crude_Data)[i] <- paste("Volume", i, sep = ".")
    }
    if (i > 3) {
        colnames(Crude_Data)[i] <- paste("Speed", i - 3, sep = ".")
    }

}

Input_Data <- as.data.frame(c(Crude_Data[, 1:2], Crude_Data[,
    4:5]), check.names = FALSE)
Output_Data <- as.data.frame(c(Crude_Data[, 3], Crude_Data[,
    6]), check.names = FALSE)

# Separate Datasets
# ------------------------------------------------------


Input_Data_Training <- Input_Data[1:537, ]
Input_Data_Testing <- Input_Data[(537 + 1):(537 + 135), ]
Input_Data_Forecast <- Input_Data[(537 + 135 + 1):nrow(Input_Data),
    ]


Output_Data_Training <- Output_Data[1:537, ]
```

```r
Output_Data_Testing <- Output_Data[(537 + 1):(537 + 135), ]
Output_Data_Forecast <- Output_Data[(537 + 135 + 1):nrow(Output_Data),
    ]

Training_Data <- cbind(Input_Data_Training, Output_Data_Training)
Validation_Data <- cbind(Input_Data_Testing, Output_Data_Testing)
Test_Data <- cbind(Input_Data_Forecast, Output_Data_Forecast)
# Training_Data <- Training_Data[, c(1,2,5,3,4,6)]
# Training_Data <- as.h2o(Training_Data);

Predictors <- names(Input_Data_Training)
Response <- names(Output_Data_Training)
```

The procedure of determining which configuration of hidden layers and number of nodes involves a for loop creating every possibility possible within the boundaries from the exercise. These boundaries were the number of hidden layers (1 to 3) and the number of nodes on each hidden layer (from 5 to 10). The package used is h2o. Keras and tensorflow were extensively tried but wouldn't work on my R installation. The package h2o has a limitation of one output per neural network. In this case, two sets of neural networks were created, one for the 'Volume 3' output and another for 'Speed 3' output.

```r
source("Run_Speed_NN.R", echo = TRUE)
```

```
##
## > Run_Speed_NN <- function(Model_File, x, y, Training_Frame,
## +     Validation_Frame) {
## +     Model_Speed <- Model_File
## +     rm(Model_File)
## +     n < .... [TRUNCATED]
```

```r
h2o.init(max_mem_size = "5g")  ## Create a new H2o session
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\PC\AppData\Local\Temp\RtmpakqSvt\file357c29176407/h2o_PC_started_from_r.out
##     C:\Users\PC\AppData\Local\Temp\RtmpakqSvt\file357c4a56381/h2o_PC_started_from_r.err
##
##
## Starting H2O JVM and connecting:  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 seconds 605 milliseconds
##     H2O cluster timezone:       Europe/Berlin
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.0.1
##     H2O cluster version age:    3 months
##     H2O cluster name:           H2O_started_from_R_PC_bnv108
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   5.00 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, Core V4
##     R Version:                  R version 4.0.2 (2020-06-22)
```

```r
h2o.no_progress()

Model_DL_Speed <- list()
Model_DL_Volume <- list()

Model_Speed <- Run_Speed_NN(Model_File = Model_DL_Speed, x = Predictors,
    y = "Speed.3", Training_Frame = Training_Data, Validation_Frame = Validation_Data)

Speed_Forecast_Evaluation <- data.frame(data.frame(matrix(data = 0,
    nrow = length(Model_Speed), ncol = 3)))
```

```r
colnames(Speed_Forecast_Evaluation) <- c("Speed3.MSE", "Speed3.RMSE",
    "Speed3.R²")
for (i in 1:length(Model_Speed)) {
    Speed_Forecast_Evaluation[i, ] <- cbind(MSE(h2o.predict(Model_Speed[[i]],
        newdata = Input_Data_Forecast %>% as.h2o()) %>% as.vector(),
        y_true = Test_Data$Speed.3), RMSE(h2o.predict(Model_Speed[[i]],
        newdata = Input_Data_Forecast %>% as.h2o()) %>% as.vector(),
        y_true = Test_Data$Speed.3), R2_Score(h2o.predict(Model_Speed[[i]],
        newdata = Input_Data_Forecast %>% as.h2o()) %>% as.vector(),
        y_true = Test_Data$Speed.3))

}
Forecast_Speed <- c()
Forecast_Speed <- h2o.predict(Model_Speed[[211]], newdata = Test_Data %>%
    as.h2o()) %>% as.vector()
```

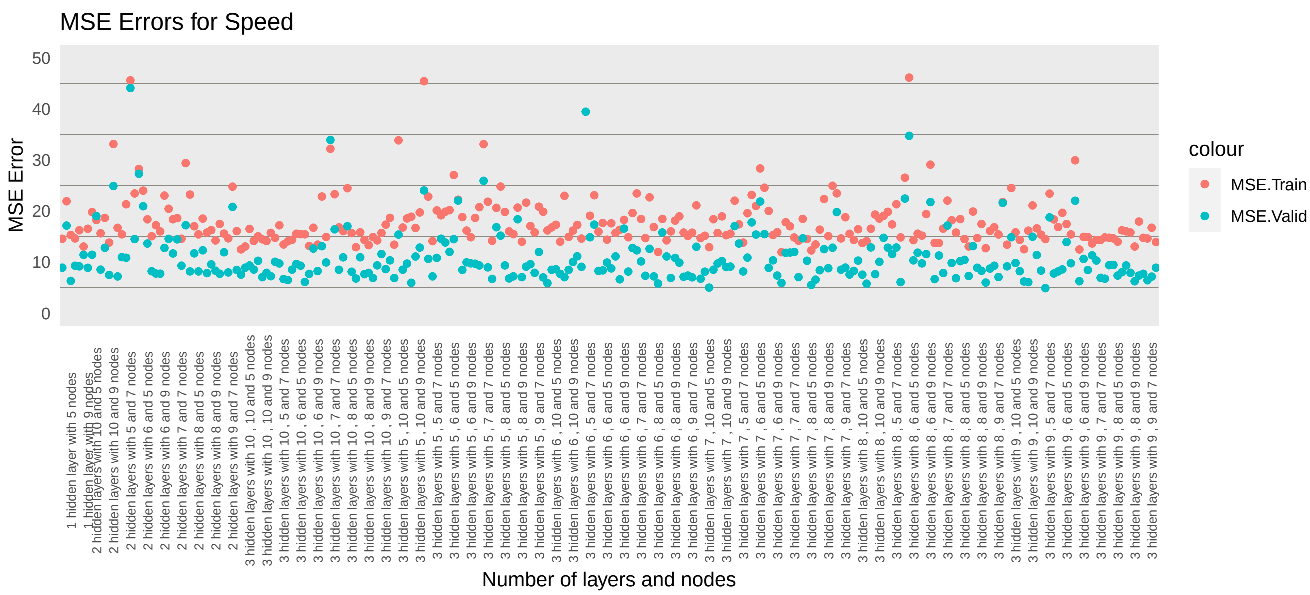
```r
source("Run_Volume_NN.R")


h2o.shutdown(prompt = FALSE)  ## Shutting the H2o session
Sys.sleep(3)  ## 3 Seconds pause
h2o.init(max_mem_size = "5g")  ## Create a new H2o session
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\PC\AppData\Local\Temp\RtmpakqSvt\file357c231970c5/h2o_PC_started_from_r.out
##     C:\Users\PC\AppData\Local\Temp\RtmpakqSvt\file357c70ff26c3/h2o_PC_started_from_r.err
##
##
## Starting H2O JVM and connecting:  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 seconds 241 milliseconds
##     H2O cluster timezone:       Europe/Berlin
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.0.1
##     H2O cluster version age:    3 months
##     H2O cluster name:           H2O_started_from_R_PC_sxb331
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   5.00 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, Core V4
##     R Version:                  R version 4.0.2 (2020-06-22)
```

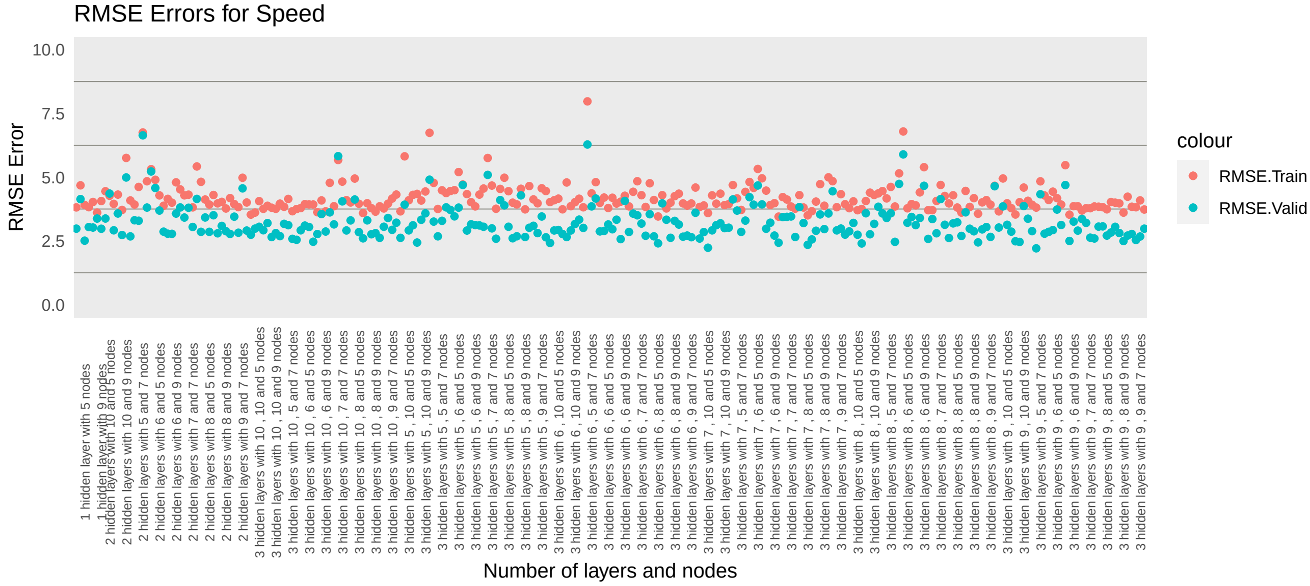
```r
h2o.no_progress()

Model_Volume <- Run_Volume_NN(Model_File = Model_DL_Volume, x = Predictors,
    y = "Volume.3", Training_Frame = Training_Data, Validation_Frame = Validation_Data)

Volume_Forecast_Evaluation <- data.frame(data.frame(matrix(data = 0,
    nrow = length(Model_Speed), ncol = 3)))
colnames(Volume_Forecast_Evaluation) <- c("Volume3.MSE", "Volume3.RMSE",
    "Volume3.R²")

for (i in 1:length(Model_Volume)) {
    Volume_Forecast_Evaluation[i, ] <- cbind(MSE(h2o.predict(Model_Volume[[i]],
        newdata = Input_Data_Forecast %>% as.h2o()) %>% as.vector(),
        y_true = Test_Data$Volume.3), RMSE(h2o.predict(Model_Volume[[i]],
        newdata = Input_Data_Forecast %>% as.h2o()) %>% as.vector(),
        y_true = Test_Data$Volume.3), R2_Score(h2o.predict(Model_Volume[[i]],
        newdata = Input_Data_Forecast %>% as.h2o()) %>% as.vector(),
        y_true = Test_Data$Volume.3))
}
Forecast_Volume <- c()

Forecast_Volume <- h2o.predict(Model_Volume[[211]], newdata = Test_Data %>%
    as.h2o()) %>% as.vector()
```

In order to compare all the models created, some metrics are compared. Mean square error (MSE) and root-mean-square error (RMSE) were used. Both values were plotted, with the model id on the horizontal axis. Model ID tells the number of layers and nodes of the neural network.

```r
Metrics_Speed3 <- data.frame(matrix(data = 0, nrow = length(Model_Speed),
    ncol = 5))
colnames(Metrics_Speed3) <- c("MSE.Train", "MSE.Valid", "RMSE.Train",
    "RMSE.Valid", "ModelID")
# Initializing the variable MSE_Speed3

for (i in 1:length(Model_Speed)) {
    Metrics_Speed3[i, ] <- c(h2o.mse(Model_Speed[[i]], valid = TRUE,
        train = TRUE), h2o.rmse(Model_Speed[[i]], valid = TRUE,
        train = TRUE), Model_Speed[[i]]@model_id)
    ## Put the MSE values of every model into the variable
    ## MSE_Speed3 for a graph
}

ggplot(data = Metrics_Speed3, mapping = aes(x = as.character(ModelID))) +
    geom_point(aes(y = as.numeric(MSE.Train), colour = "MSE.Train")) +
    geom_point(aes(y = as.numeric(MSE.Valid), colour = "MSE.Valid")) +
    ylim(0, 50) + theme(axis.ticks = element_line(size = 0.2,
    linetype = "blank"), panel.grid.major = element_line(linetype = "blank"),
    panel.grid.minor = element_line(colour = "ivory4"), axis.text.x = element_text(angle = 90,
        size = 7)) + xlab("Number of layers and nodes") + ylab("MSE Error") +
    scale_x_discrete(breaks = Metrics_Speed3$ModelID[c(T, F,
        F, F)]) + ggtitle("MSE Errors for Speed")
```

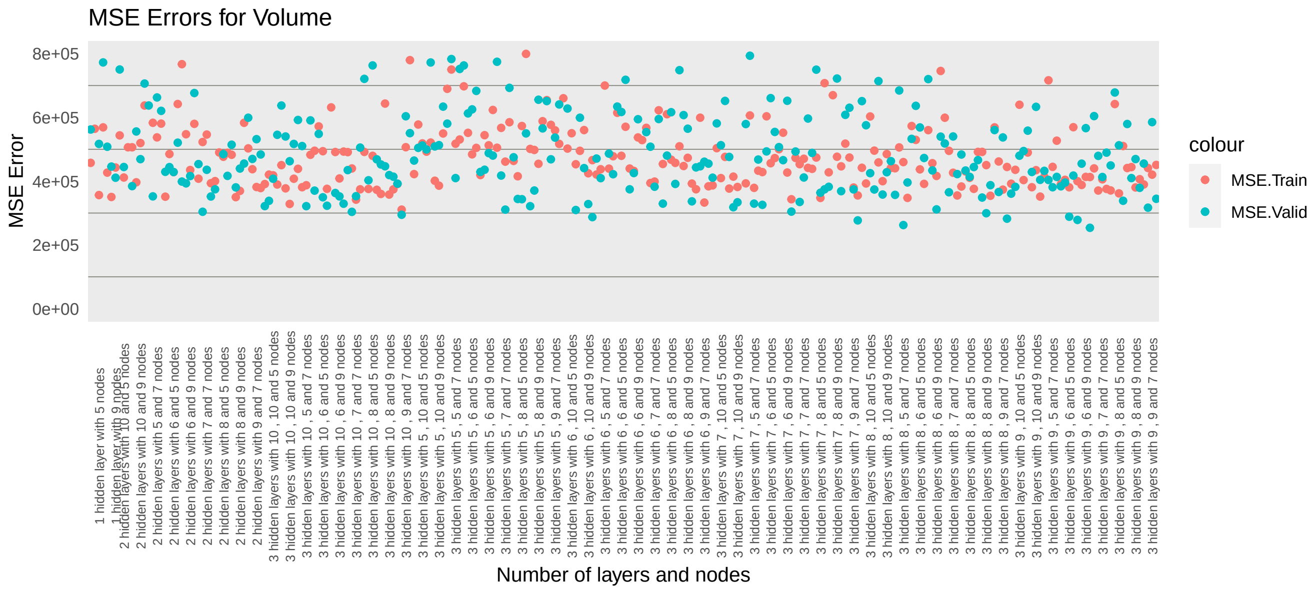



```r
ggplot(data = Metrics_Speed3, mapping = aes(x = as.character(ModelID))) +
    geom_point(aes(y = as.numeric(RMSE.Train), colour = "RMSE.Train")) +
    geom_point(aes(y = as.numeric(RMSE.Valid), colour = "RMSE.Valid")) +
    ylim(0, 10) + theme(axis.ticks = element_line(size = 0.2,
    linetype = "blank"), panel.grid.major = element_line(linetype = "blank"),
    panel.grid.minor = element_line(colour = "ivory4"), axis.text.x = element_text(angle = 90,
        size = 7)) + xlab("Number of layers and nodes") + ylab("RMSE Error") +
    scale_x_discrete(breaks = Metrics_Speed3$ModelID[c(T, F,
        F, F)]) + ggtitle("RMSE Errors for Speed")
```

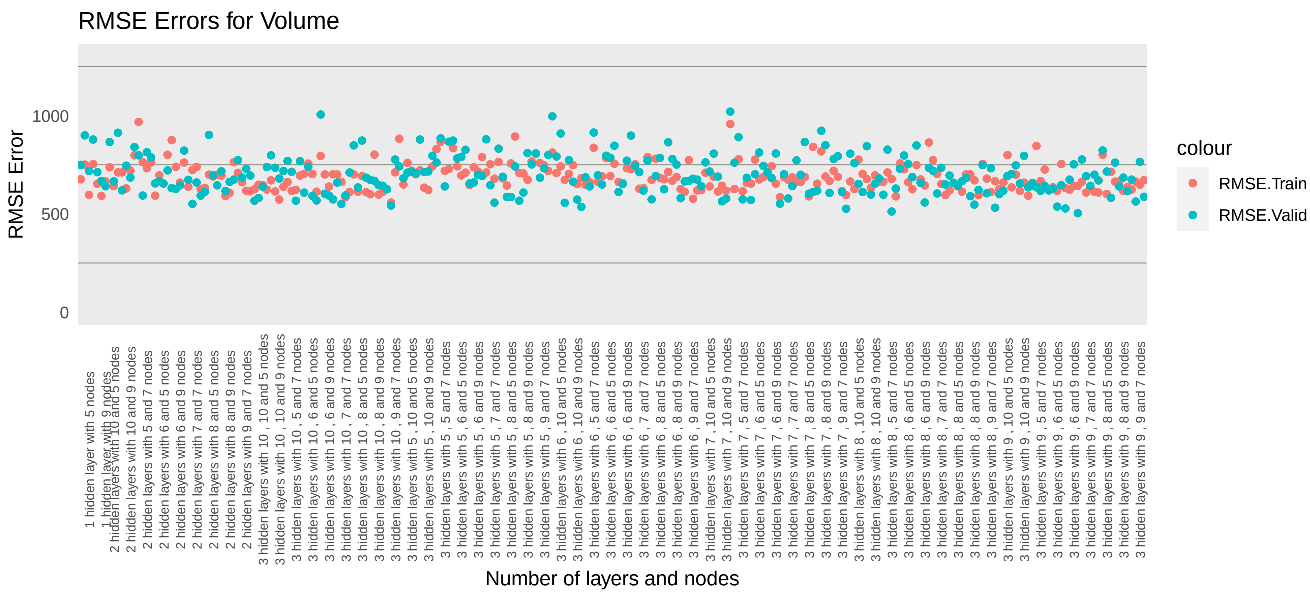


The same steps are taken for the Volume in position 3 as output.

```
## Metrics for the Volume3 neural networks

Metrics_Volume3 <- data.frame(matrix(data = 0, nrow = length(Model_Volume),
    ncol = 5))
colnames(Metrics_Volume3) <- c("MSE.Train", "MSE.Valid", "RMSE.Train",
    "RMSE.Valid", "ModelID")
# Initializing the variable MSE_Speed3

for (i in 1:length(Model_Volume)) {
    Metrics_Volume3[i, ] <- c(h2o.mse(Model_Volume[[i]], valid = TRUE,
        train = TRUE), h2o.rmse(Model_Volume[[i]], valid = TRUE,
        train = TRUE), Model_Volume[[i]]@model_id)
    ## Put the MSE values of every model into the variable
    ## MSE_Speed3 for a graph
}


ggplot(data = Metrics_Volume3, mapping = aes(x = as.character(ModelID))) +
    geom_point(aes(y = as.numeric(MSE.Train), colour = "MSE.Train")) +
    geom_point(aes(y = as.numeric(MSE.Valid), colour = "MSE.Valid")) +
    ylim(0, 8e+05) + theme(axis.ticks = element_line(size = 0.2,
    linetype = "blank"), panel.grid.major = element_line(linetype = "blank"),
    panel.grid.minor = element_line(colour = "ivory4"), axis.text.x = element_text(angle = 90,
        size = 7)) + xlab("Number of layers and nodes") + ylab("MSE Error") +
    scale_x_discrete(breaks = Metrics_Volume3$ModelID[c(T, F,
        F, F)]) + ggtitle("MSE Errors for Volume")
```



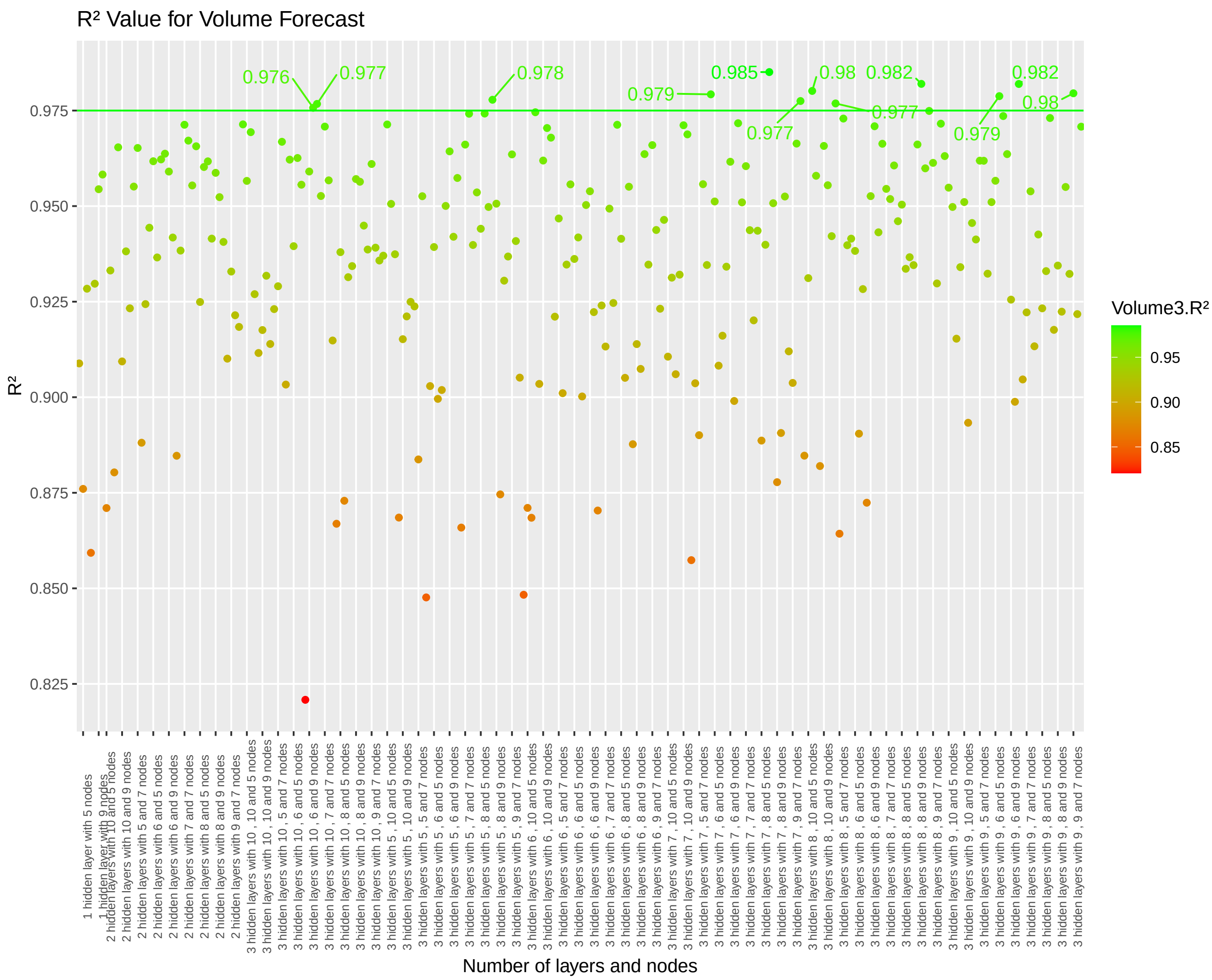

```
ggplot(data = Metrics_Volume3, mapping = aes(x = as.character(ModelID))) +
    geom_point(aes(y = as.numeric(RMSE.Train), colour = "RMSE.Train")) +
    geom_point(aes(y = as.numeric(RMSE.Valid), colour = "RMSE.Valid")) +
    ylim(0, 1300) + theme(axis.ticks = element_line(size = 0.2,
    linetype = "blank"), panel.grid.major = element_line(linetype = "blank"),
    panel.grid.minor = element_line(colour = "ivory4"), axis.text.x = element_text(angle = 90,
        size = 7)) + xlab("Number of layers and nodes") + ylab("RMSE Error") +
    scale_x_discrete(breaks = Metrics_Volume3$ModelID[c(T, F,
        F, F)]) + ggtitle("RMSE Errors for Volume")
```



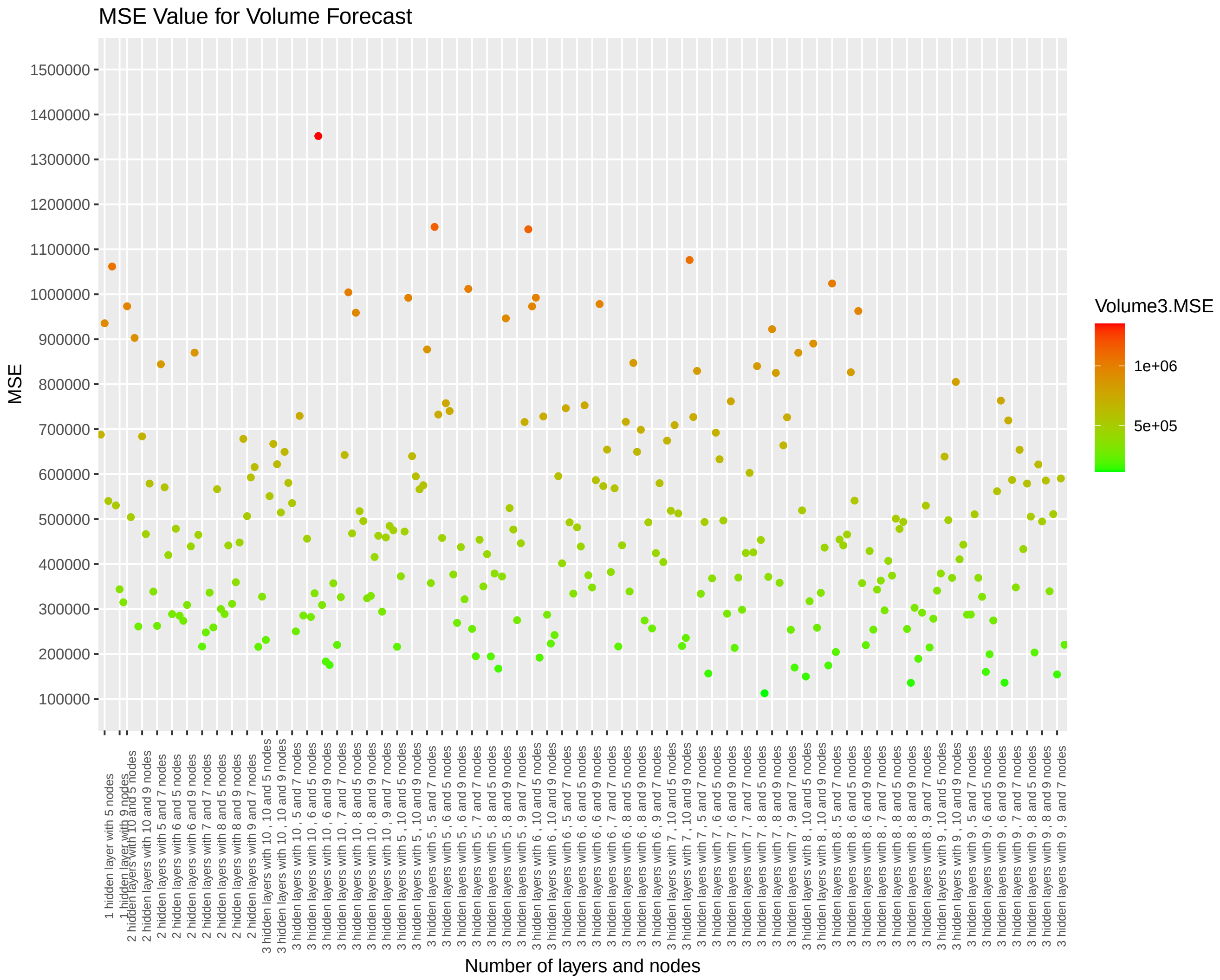
It can be seen in the graphs that the performance of the networks don't change munch according to their hidden layer number and number of nodes. The three metrics to evaluate the forecast's accuracy are MSE, RMSE and R-Squared.

```
rownames(Volume_Forecast_Evaluation) <- Metrics_Volume3$ModelID
rownames(Speed_Forecast_Evaluation) <- Metrics_Speed3$ModelID

ggplot(data = Volume_Forecast_Evaluation, mapping = aes(y = Volume3.R²,
    x = rownames(Volume_Forecast_Evaluation), color = Volume3.R²,
    label = ifelse(Volume3.R² >= 0.975, base::round(Volume3.R²,
        3), ""))) + geom_point() + scale_color_gradient(low = "red",
    high = "green") + theme(axis.ticks = element_line(linetype = "solid"),
    panel.grid.major = element_line(linetype = "solid"), panel.grid.minor = element_line(colour = "ivory4",
        linetype = "blank"), axis.text.x = element_text(angle = 90,
        size = 7), axis.ticks.x = element_line(linetype = "solid")) +
```

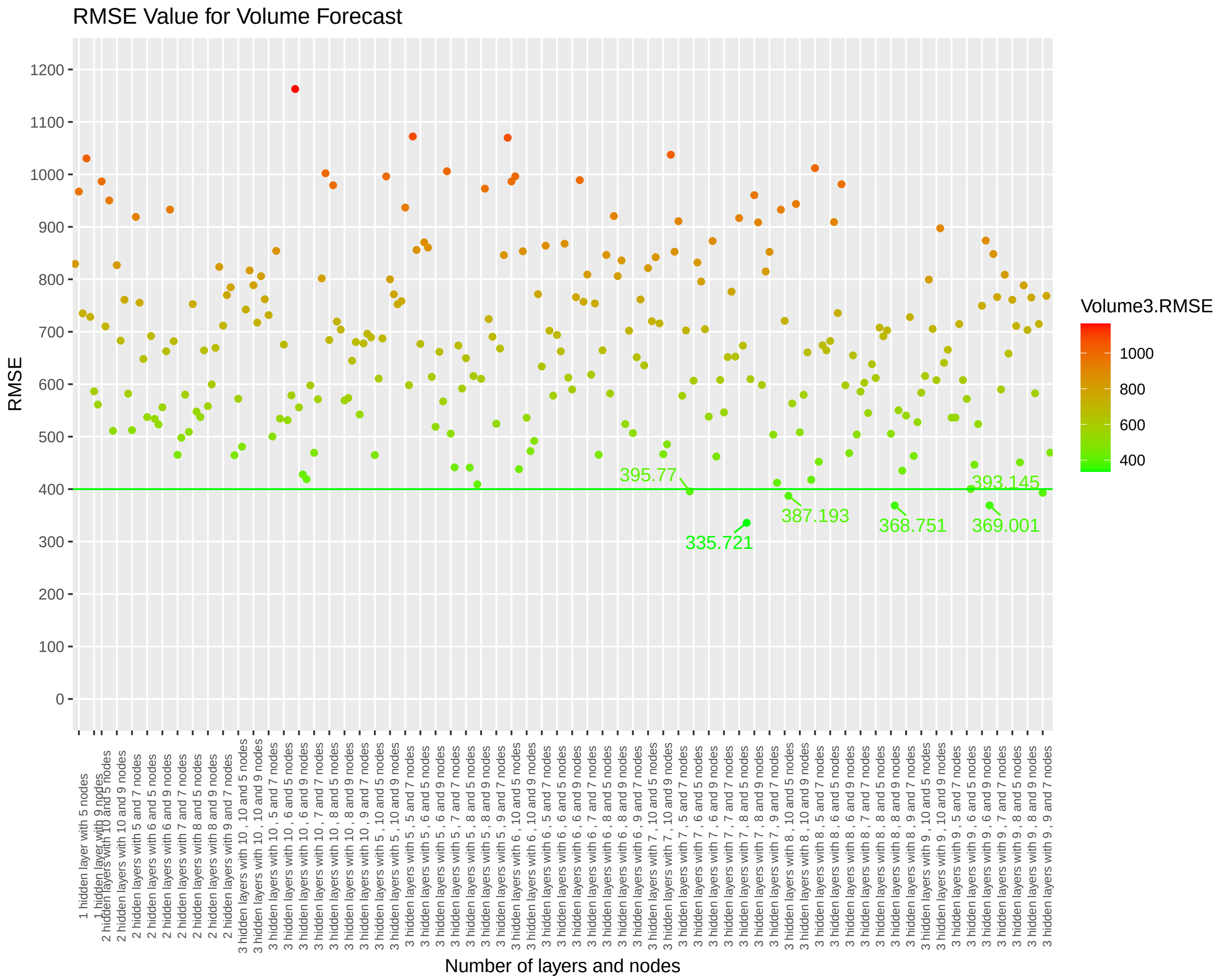
```
    xlab("Number of layers and nodes") + ylab("R²") + scale_x_discrete(breaks = Metrics_Volume3$ModelID[c(T,
    F, F, F)]) + ggtitle("R² Value for Volume Forecast") + scale_y_continuous(breaks = seq(0.8,
    1, 0.025)) + geom_text_repel(box.padding = 1, min.segment.length = 0) +
    geom_hline(yintercept = 0.975, color = "green")
```




```
ggplot(data = Volume_Forecast_Evaluation, mapping = aes(y = Volume3.MSE,
    x = rownames(Volume_Forecast_Evaluation), color = Volume3.MSE)) +
    geom_point() + scale_color_gradient(low = "green", high = "red") +
    theme(axis.ticks = element_line(linetype = "solid"), panel.grid.major = element_line(linetype = "solid"),
        panel.grid.minor = element_line(colour = "ivory4", linetype = "blank"),
        axis.text.x = element_text(angle = 90, size = 7), axis.ticks.x = element_line(linetype = "solid")) +
    xlab("Number of layers and nodes") + ylab("MSE") + scale_x_discrete(breaks = Metrics_Volume3$ModelID[c(T,
    F, F, F)]) + ggtitle("MSE Value for Volume Forecast") + scale_y_continuous(n.breaks = 12,
    limits = c(1e+05, 1500000))
```



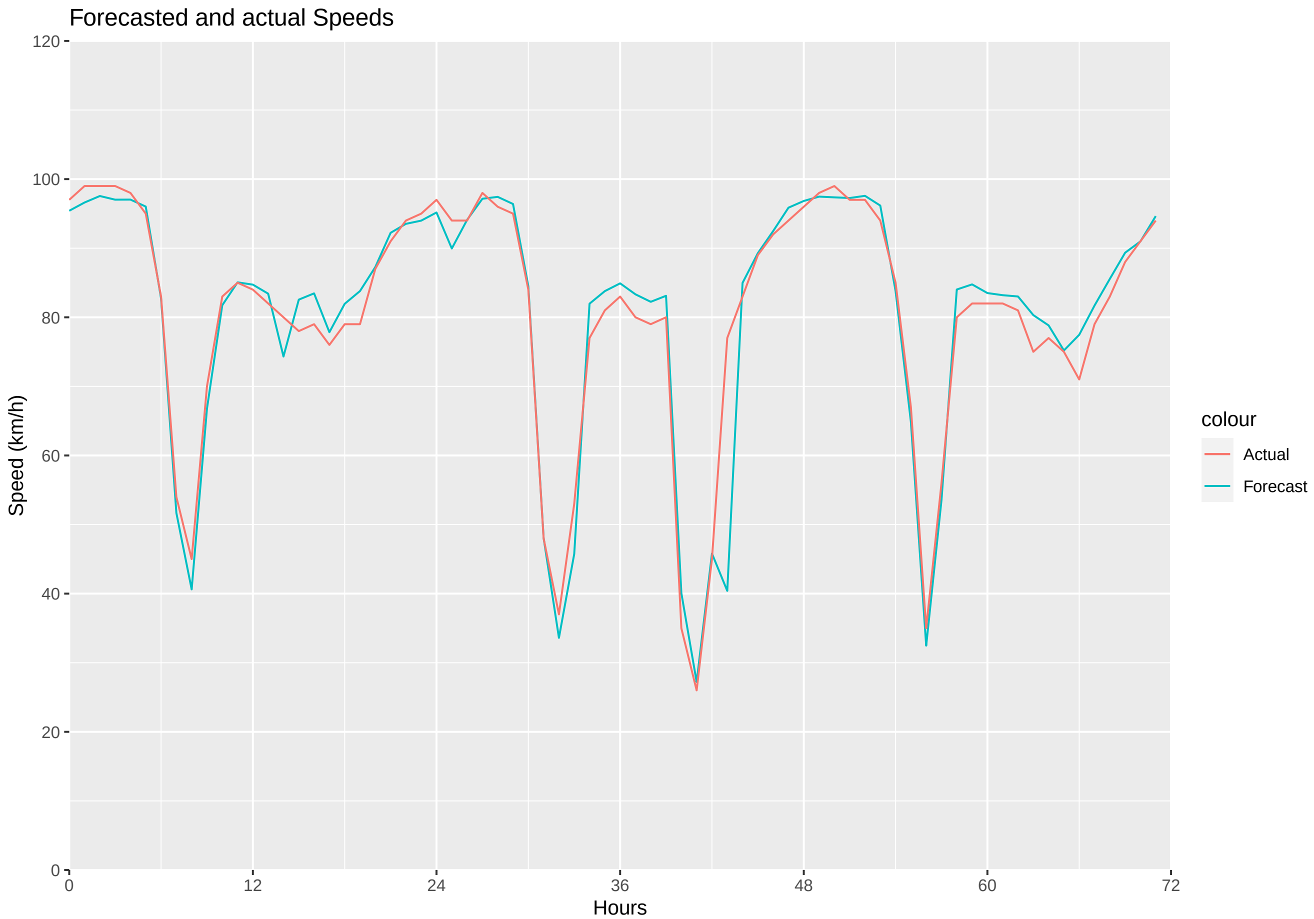

```
ggplot(data = Volume_Forecast_Evaluation, mapping = aes(y = Volume3.RMSE,
    x = rownames(Volume_Forecast_Evaluation), color = Volume3.RMSE,
    label = ifelse(Volume3.RMSE <= 400, base::round(Volume3.RMSE,
        3), ""))) + geom_point() + scale_color_gradient(low = "green",
    high = "red") + theme(axis.ticks = element_line(linetype = "solid"),
    panel.grid.major = element_line(linetype = "solid"), panel.grid.minor = element_line(colour = "ivory4",
        linetype = "blank"), axis.text.x = element_text(angle = 90,
        size = 7), axis.ticks.x = element_line(linetype = "solid")) +
    xlab("Number of layers and nodes") + ylab("RMSE") + scale_x_discrete(breaks = Metrics_Volume3$ModelID[c(T,
    F, F, F)]) + ggtitle("RMSE Value for Volume Forecast") +
    scale_y_continuous(n.breaks = 12, limits = c(0, 1200)) +
    geom_hline(yintercept = 400, color = "green") + geom_text_repel(nudge_y = -1,
    box.padding = 0.5, min.segment.length = 0)
```



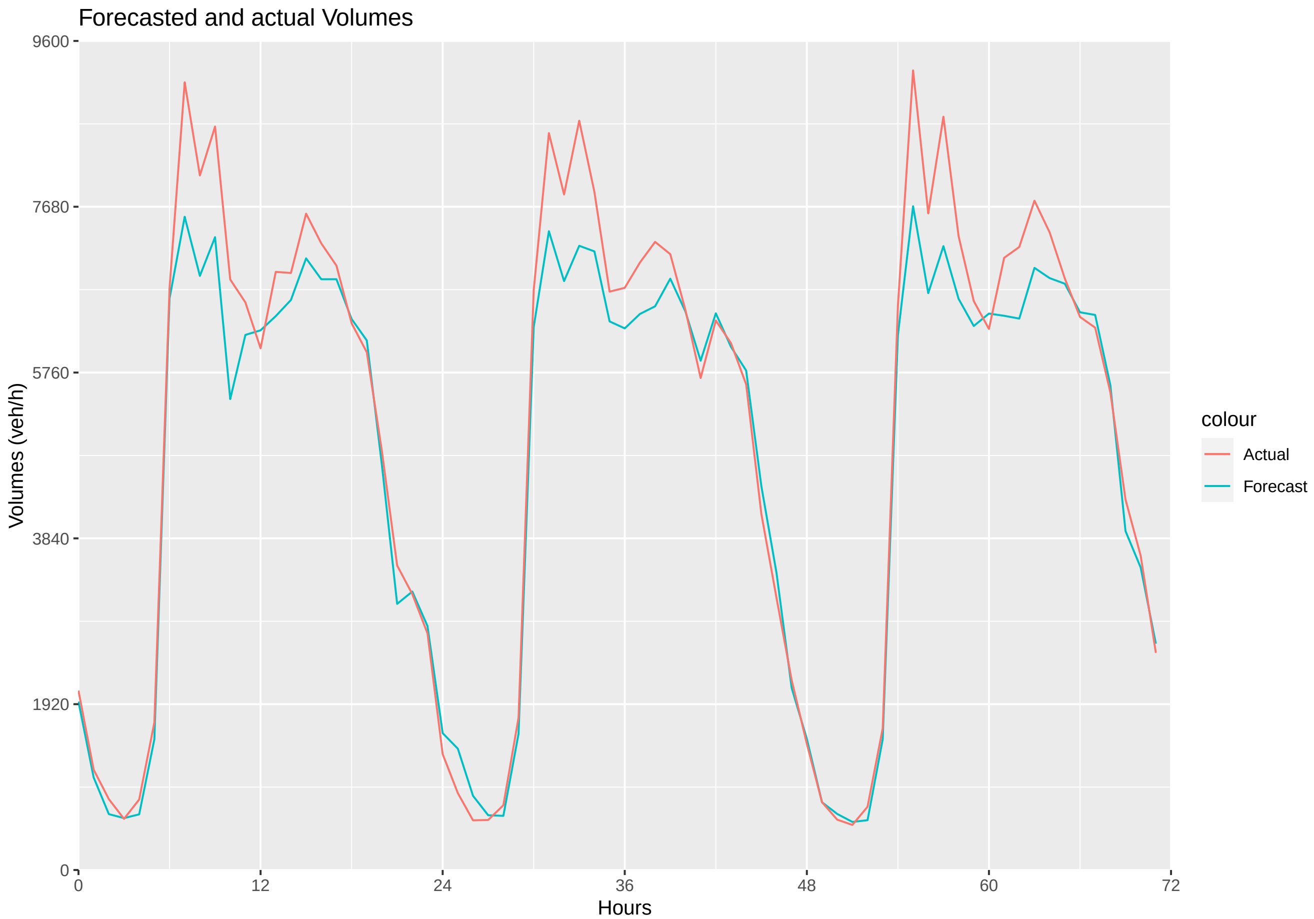
The model with 3 hidden layers, with 9, 9 and 5 layers presented good performance on the 3 tests. This model was chosen to perform the forecast with the 3 day data. This model was the 211th model to be performed.

```
Forecast_Volume <- as.data.frame(cbind(Forecast_Volume, Output_Data_Forecast$Volume.3))
colnames(Forecast_Volume) <- c("Forecasted Volume", "Actual Volume")

Forecast_Speed <- as.data.frame(cbind(Forecast_Speed, Output_Data_Forecast$Speed.3))
colnames(Forecast_Speed) <- c("Forecasted Speed", "Actual Speed")
```

```
ggplot(data = Forecast_Speed, aes(x = 0:71)) + geom_line(aes(y = `Forecasted Speed`,
    color = "Forecast")) + geom_line(aes(y = `Actual Speed`,
    color = "Actual")) + ggtitle("Forecasted and actual Speeds") +
    xlab("Hours") + ylab("Speed (km/h)") + scale_x_continuous(breaks = seq(0,
    72, 12), limits = c(0, 72), expand = c(0, 0)) + scale_y_continuous(limits = c(0,
    120), expand = c(0, 0), breaks = seq(0, 120, 20))
```




```r
ggplot(data = Forecast_Volume, aes(x = 0:71)) + geom_line(aes(y = `Forecasted Volume`,
    color = "Forecast")) + geom_line(aes(y = `Actual Volume`,
    color = "Actual")) + ggtitle("Forecasted and actual Volumes") +
    xlab("Hours") + ylab("Volumes (veh/h)") + scale_x_continuous(breaks = seq(0,
    72, 12), limits = c(0, 72), expand = c(0, 0)) + scale_y_continuous(limits = c(0,
    9600), expand = c(0, 0), breaks = seq(0, 9600, 1920))
```



Unfortunately the H2o package doesn't provide a function to easily plot the neural network.

Exercise 3

07 July 2020

(a) How many regressor variables are in this model?

Number of Variables = 3

(b) If the error sum of squares is 307 and there are 15 observations, what is the estimate of σ^2?

$$\sigma^2 = \frac{SSe}{N-1}$$

```
SSe<- 307
N <- 15

Variance <- SSe/(N - 1)
print(Variance)
```

```
## [1] 21.92857
```

(c) What is the standard error of the regression coefficient $\hat{\beta}_1$?

Variance σ^2 was calculated on question B, with value 21.929. The equation to discover the error of the regression coefficient is

$$SE(\hat{\beta}_1) = \sqrt{\frac{\hat{\sigma}^2}{\sum_{i=1}^{n}(x_i - \overline{x})^2}}$$

Since the value on position [3,3] on the matrix is represents

$$\sum_{i=1}^{n}(x_i - \overline{x})^2$$

, then the $SE(\hat{\beta}_1)$ is $\sigma^2 \times 0.0009108 = 0.0199725$

Exercise 5

Arthur Junges Schmidt

08 July 2020

(a) Linear regression This model would have a high bias and a low variance. The high bias occur due to the oversimplification of the regression (underfitting) by using a linear regression for a model with 4 predictors. The variance is lower because it does a better job generalizing for other data sets.

(b) Polynomial regression with degree 3 This model would have a medium value for both bias and variance. While it's more complex and does a better job decreasing the fitting error than linear regression, it still lacks complexity. However, with 3 degrees it would present more variance as it would increase error when utilizing other data sets.

(c) Polynomial regression with degree 10 This model would present a small bias and a high variance. While it would do a good job fitting very well the training data, it may overfit it. By overfitting the data, the variance of the random error would increase.

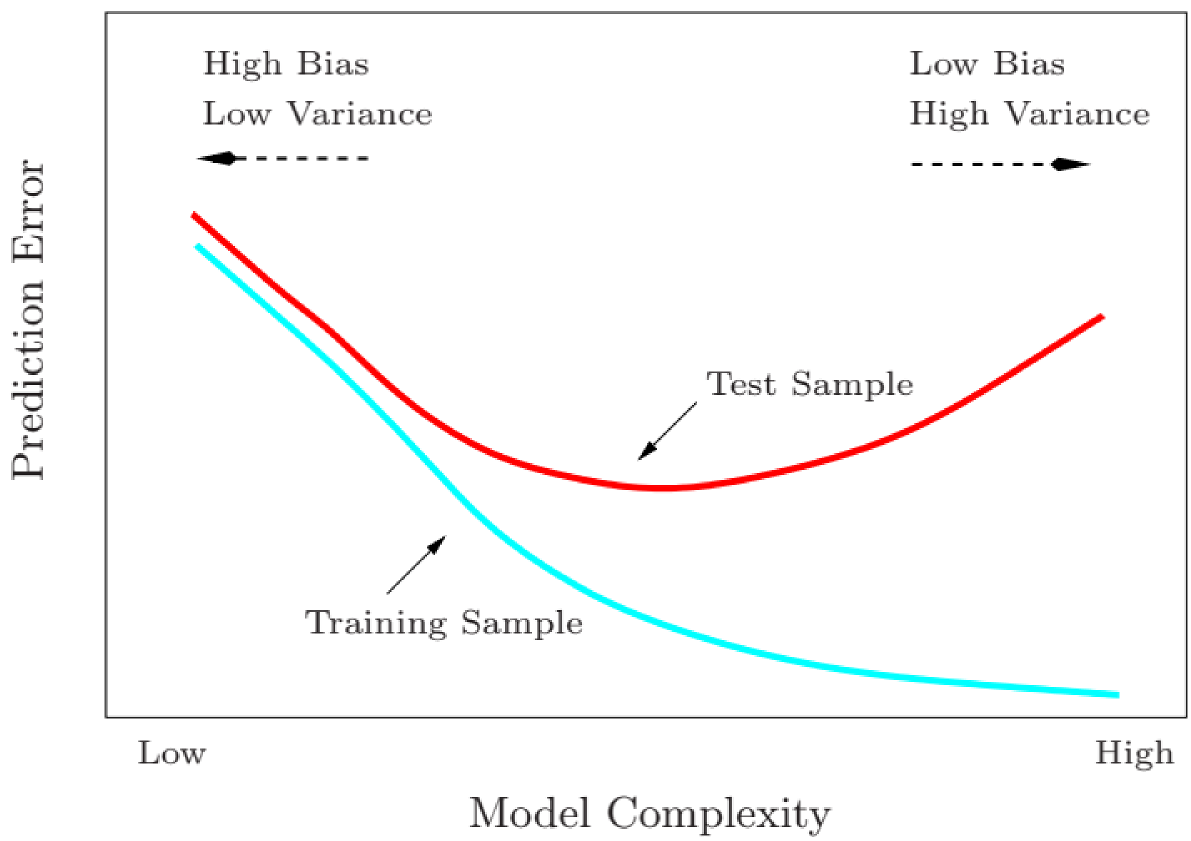


Figure 1: Test and training error as a function of model complexity. Hastie, T., Friedman, J., & Tisbshirani, R. (2018). The Elements of statistical learning: Data mining, inference, and prediction. New York: Springer.

Exercise 6

Arthur Junges Schmidt

08 July 2020

The data which can be found in a separate spreadsheet provides the highway gasoline mileage test results for 2005 model year vehicles from DaimlerChrysler. (1) Fit a multiple linear regression model to these data to estimate gasoline mileage that uses the following regressors: cid, rhp, etw, cmp, axle, n/v

```
Linear_Model <- lm(formula = mpg ~ cid + rhp + etw + cmp + axle +
                     `n/v`,
                   data = Original_Data);
summary(Linear_Model)
```

```
##
## Call:
## lm(formula = mpg ~ cid + rhp + etw + cmp + axle + `n/v`, data = Original_Data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.0501 -0.8477  0.2360  1.0896  2.8193
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 49.9039998 19.6652426   2.538  0.02368 *
## cid         -0.0104474  0.0233788  -0.447  0.66180
## rhp         -0.0012042  0.0163061  -0.074  0.94217
## etw         -0.0032364  0.0009459  -3.421  0.00413 **
## cmp          0.2924277  1.7647364   0.166  0.87076
## axle        -3.8553646  1.3286464  -2.902  0.01160 *
## `n/v`        0.1897094  0.2729740   0.695  0.49845
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.228 on 14 degrees of freedom
## Multiple R-squared:  0.8933, Adjusted R-squared:  0.8476
## F-statistic: 19.53 on 6 and 14 DF,  p-value: 4.664e-06
```

(2) Estimate σ^2 and the standard errors of the regression coefficients.

σ^2 values:

```
##  (Intercept)          cid          rhp          etw          cmp         axle
## 3.867218e+02 5.465676e-04 2.658901e-04 8.947518e-07 3.114294e+00 1.765301e+00
##        `n/v`
## 7.451482e-02
```

Standard errors:

```
## (Intercept)          cid          rhp          etw          cmp         axle
## 1.966524e+01 2.337879e-02 1.630614e-02 9.459132e-04 1.764736e+00 1.328646e+00
##        `n/v`
## 2.729740e-01
```

(3) Test for significance of regression using $\alpha = 0.05$. What conclusions can you draw? Only the intercept and the 'etw' and 'axle' variables have a significance of 95% or higher. This leads to the conclusion that, with 95% certainty, only the axle ration and the equivalent test weight variables are significant to explain the mileage of the vehicles on the data set.

(4) Find the t-test statistic for each regressor. Using $\alpha = 0.05$, what conclusions can you draw? Does each regressor contribute to the model?

T-test statistic for each regressor:

```
## (Intercept)         cid         rhp         etw         cmp        axle
##  2.53767527 -0.44687602 -0.07385193 -3.42145672  0.16570617 -2.90172361
##       `n/v`
##  0.69497223
```

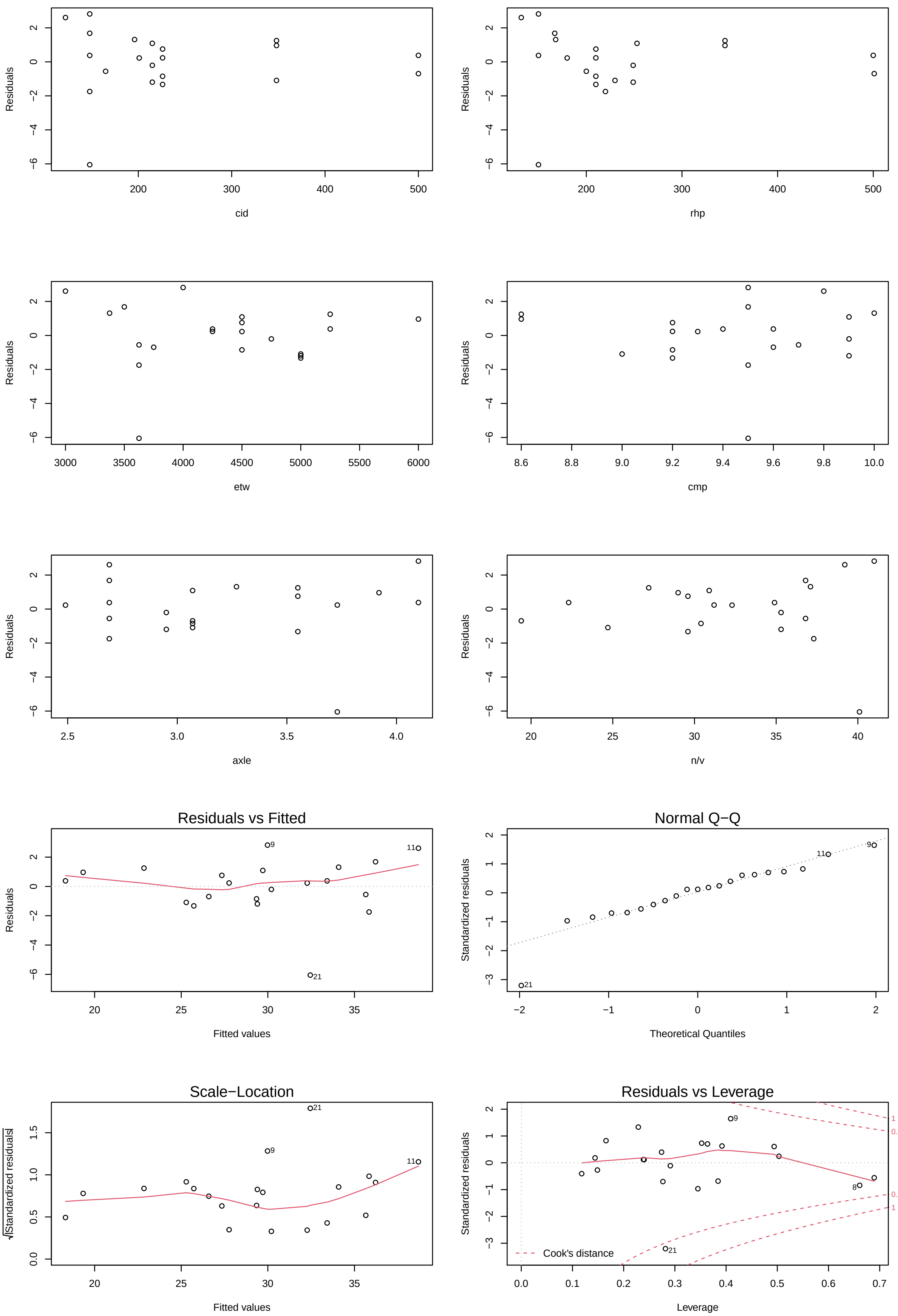
As on the question before, the only 2 variables that have a significant t-test are 'etw' and 'axle'. Not all the regressor contributes to the model since some of them add more error into it.

(5) Find 99% confidence intervals on the regression coefficients.

```
##                    0.5 %       99.5 %
## (Intercept) -8.636334830  1.084443e+02
## cid         -0.080042386  5.914755e-02
## rhp         -0.049745045  4.733657e-02
## etw         -0.006052236 -4.205662e-04
## cmp         -4.960914978  5.545770e+00
## axle        -7.810535975  9.980675e-02
## `n/v`       -0.622891389  1.002310e+00
```

(6) Plot residuals versus $\hat{Y}$ and versus each regressor. Discuss these residual plots.

```r
par(mfrow = c(5,2))
plot(x = Linear_Model$model$cid, y = resid(Linear_Model), xlab = "cid", ylab = "Residuals")
plot(x = Linear_Model$model$rhp, y = resid(Linear_Model), xlab = "rhp", ylab = "Residuals")
plot(x = Linear_Model$model$etw, y = resid(Linear_Model), xlab = "etw", ylab = "Residuals")
plot(x = Linear_Model$model$cmp, y = resid(Linear_Model), xlab = "cmp", ylab = "Residuals")
plot(x = Linear_Model$model$axle, y = resid(Linear_Model), xlab = "axle", ylab = "Residuals")
plot(x = Linear_Model$model$`n/v`, y = resid(Linear_Model), xlab = "n/v", ylab = "Residuals")
plot(Linear_Model)
```


Residuals
cid
Residuals
rhp
Residuals
etw
Residuals
cmp
Residuals
axle
Residuals
n/v
Residuals vs Fitted
Residuals
Fitted values
Normal Q−Q
Standardized residuals
Theoretical Quantiles
Scale−Location
√|Standardized residuals|
Fitted values
Residuals vs Leverage
Standardized residuals
Cook's distance
Leverage