# Neural Network

*J. Hartsfield*

*March 17, 2017*

### Neural Network examples using neuralnet package

Neural network with no hidden layers is just regression

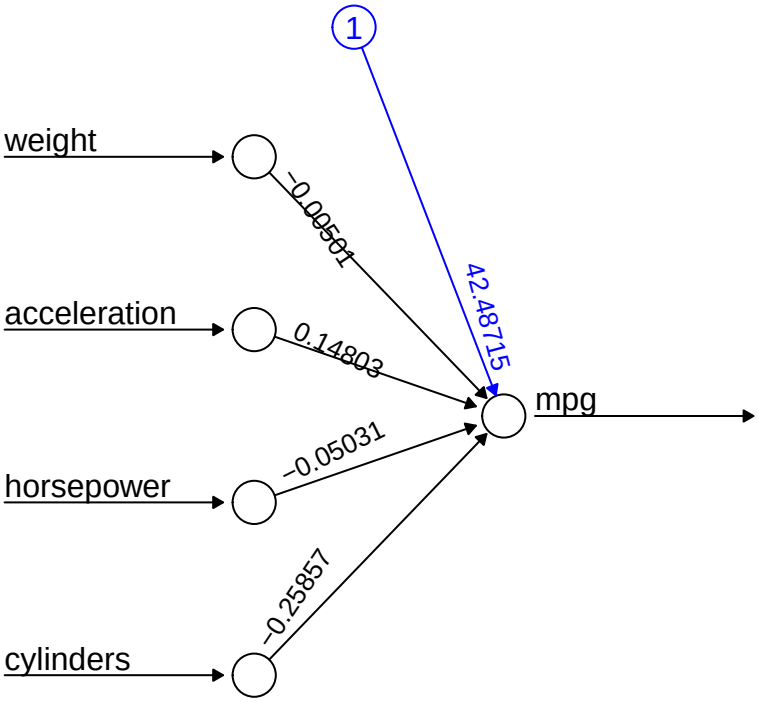
```
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 3.3.3
```

```
load("Auto.rda")
n = length(Auto$mpg)
set.seed(1234)
Z = sample(n, 200)
Auto.train = Auto[Z,]
Auto.test = Auto[-Z,]
nn0 = neuralnet(mpg ~ weight + acceleration + horsepower + cylinders, data=Auto.train, hidden=0)
summary(nn0)
```

```
##                     Length Class      Mode
## call                  4    -none-     call
## response            200    -none-     numeric
## covariate           800    -none-     numeric
## model.list            2    -none-     list
## err.fct               1    -none-     function
## act.fct               1    -none-     function
## linear.output         1    -none-     logical
## data                  9    data.frame list
## net.result            1    -none-     list
## weights               1    -none-     list
## startweights          1    -none-     list
## generalized.weights   1    -none-     list
## result.matrix         8    -none-     numeric
```

```
plot(nn0, rep="best")
```




Error: 1829.965357  Steps: 6697

Compare this to linear regression:

```
linMod = lm(mpg ~ weight + acceleration + horsepower + cylinders, data=Auto.train)
summary(linMod)
```

```
##
## Call:
## lm(formula = mpg ~ weight + acceleration + horsepower + cylinders,
##     data = Auto.train)
##
## Residuals:
##         Min          1Q      Median          3Q         Max
## -10.5315078  -2.5910769  -0.3724999   2.4736324  16.3477169
##
## Coefficients:
##                  Estimate   Std. Error  t value               Pr(>|t|)
## (Intercept)  42.487281370  3.555224343 11.95066 < 0.000000000000000222 ***
## weight       -0.005014105  0.001062895 -4.71741            0.000004546 ***
## acceleration  0.148018374  0.181142033  0.81714               0.414846
## horsepower   -0.050314865  0.022368342 -2.24938               0.025606 *
## cylinders    -0.258574594  0.430455304 -0.60070               0.548737
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.332306 on 195 degrees of freedom
## Multiple R-squared:  0.697537,   Adjusted R-squared:  0.6913326
## F-statistic: 112.4267 on 4 and 195 DF,  p-value: < 0.00000000000000022204
```

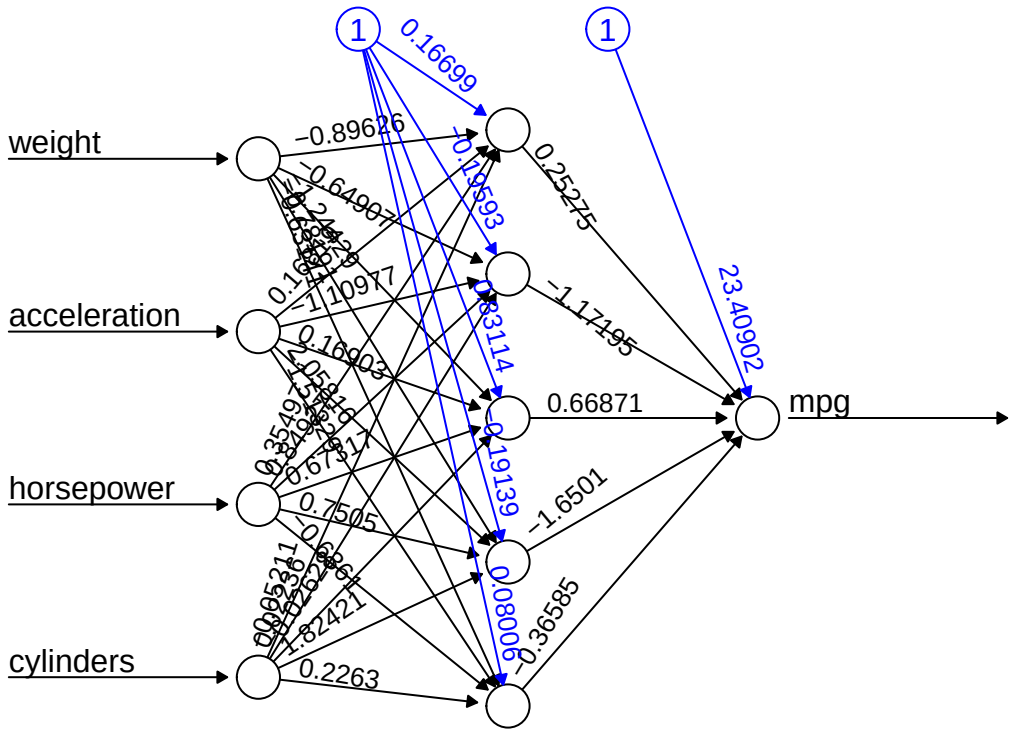
```r
# Predicted data from lm
pr.lm <- predict(linMod,Auto.test)

# Test MSE
MSE.lm <- sum((pr.lm - Auto.test$mpg)^2)/nrow(Auto.test)
```

Now we introduce a hidden layer that has 5 nodes.

```r
nn5 =neuralnet(mpg ~ weight + acceleration + horsepower + cylinders, data=Auto.train, hidden=5)
summary(nn5)
```

```
##                     Length Class      Mode
## call                  4    -none-     call
## response            200    -none-     numeric
## covariate           800    -none-     numeric
## model.list            2    -none-     list
## err.fct               1    -none-     function
## act.fct               1    -none-     function
## linear.output         1    -none-     logical
## data                  9    data.frame list
## net.result            1    -none-     list
## weights               1    -none-     list
## startweights          1    -none-     list
## generalized.weights   1    -none-     list
## result.matrix        34    -none-     numeric
```

```r
#nn5
plot(nn5, rep="best")
```



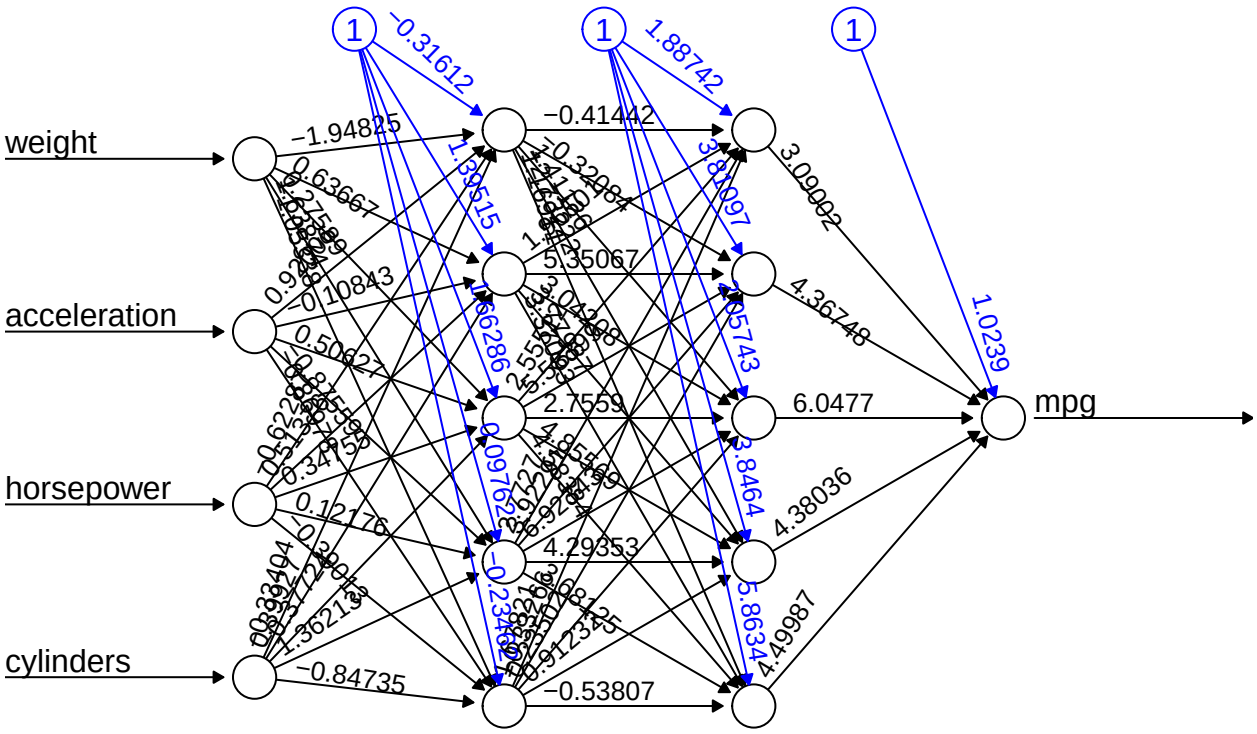

Error: 6050.2119  Steps: 252

or two hidden layers with 5 nodes each

```
nn55 =neuralnet(mpg ~ weight + acceleration + horsepower + cylinders, data=Auto.train, hidden=c(5,5))
summary(nn55)
```

```
##                     Length Class      Mode
## call                   4   -none-     call
## response             200   -none-     numeric
## covariate            800   -none-     numeric
## model.list             2   -none-     list
## err.fct                1   -none-     function
## act.fct                1   -none-     function
## linear.output          1   -none-     logical
## data                   9   data.frame list
## net.result             1   -none-     list
## weights                1   -none-     list
## startweights           1   -none-     list
## generalized.weights    1   -none-     list
## result.matrix         64   -none-     numeric
```

```
#nn55
plot(nn55, rep="best")
```




Which Neural Network has the best prediction?

```
Predict0 = compute( nn0,  subset(Auto.test, select=c(weight, acceleration, horsepower, cylinders) ) )

# This prediction consists of X-variables "neurons" and predicted Y-variable "net.result".

names(Predict0)
```

```
## [1] "neurons"    "net.result"
```

```
mean( (Auto.test$mpg - Predict0$net.result)^2 )
```

```
## [1] 17.8324906
```

```
# Prediction MSE of this ANN is 18.83.

Predict5 = compute( nn5, subset(Auto.test, select=c(weight,acceleration,horsepower,cylinders) ) )
mean( (Auto.test$mpg - Predict5$net.result)^2 )
```

```
## [1] 61.03699646
```

```
# Its 3-node competitor has a much higher prediction MSE.

Predict55 = compute( nn55, subset(Auto.test, select=c(weight,acceleration,horsepower,cylinders) ) )
mean( (Auto.test$mpg - Predict55$net.result)^2 )
```

```
## [1] 61.03699901
```

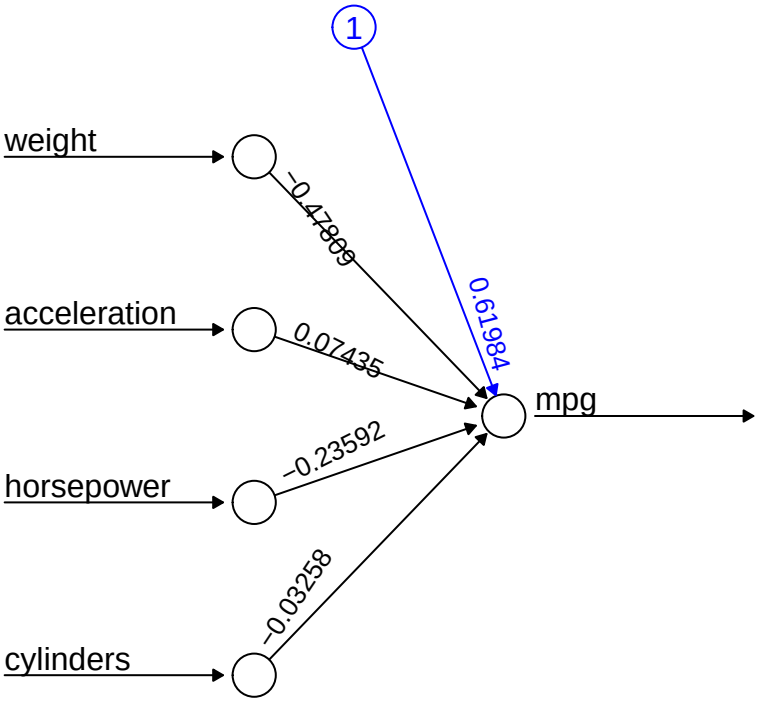
This more complicated network should give us better results, not worse!! What is wrong?

Let's try scaling the data before we train our neural network.

```
AutoSc <- Auto
AutoSc$mpg <- scale(Auto$mpg, center = min(Auto$mpg), scale = max(Auto$mpg)-min(Auto$mpg))
AutoSc$weight <- scale(Auto$weight, center = min(Auto$weight), scale = max(Auto$weight)-min(Auto$weight
AutoSc$acceleration <- scale(Auto$acceleration, center = min(Auto$acceleration), scale = max(Auto$accele
AutoSc$horsepower <- scale(Auto$horsepower, center = min(Auto$horsepower), scale = max(Auto$horsepower)-
AutoSc$cylinders <- scale(Auto$cylinders, center = min(Auto$cylinders), scale = max(Auto$cylinders)-min

trainData <- AutoSc[Z,]
testData <- AutoSc[-Z,]
nn0 = neuralnet(mpg ~ weight + acceleration + horsepower + cylinders, data=trainData, hidden=0)
summary(nn0)
```

```
##                   Length Class      Mode
## call                 4   -none-     call
## response           200   -none-     numeric
## covariate          800   -none-     numeric
## model.list           2   -none-     list
## err.fct              1   -none-     function
## act.fct              1   -none-     function
## linear.output        1   -none-     logical
## data                 9   data.frame list
## net.result           1   -none-     list
## weights              1   -none-     list
## startweights         1   -none-     list
## generalized.weights  1   -none-     list
## result.matrix        8   -none-     numeric
```

```
plot(nn0, rep="best")
```

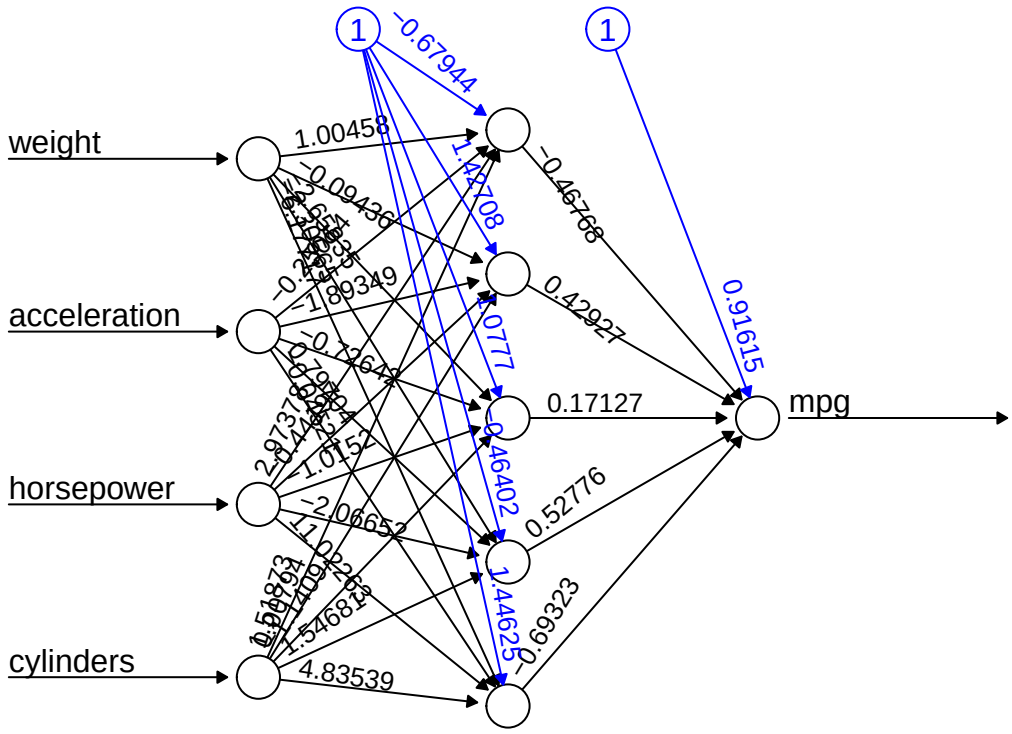



Error: 1.294473  Steps: 526

```
pr.nn0 = compute( nn0,  subset(testData, select=c(weight, acceleration, horsepower, cylinders) ) )

nn5 = neuralnet(mpg ~ weight + acceleration + horsepower + cylinders, data=trainData, hidden=5)
summary(nn5)
```

```
##                   Length Class      Mode
## call                   4 -none-     call
## response             200 -none-     numeric
## covariate            800 -none-     numeric
## model.list             2 -none-     list
## err.fct                1 -none-     function
## act.fct                1 -none-     function
## linear.output          1 -none-     logical
## data                   9 data.frame list
## net.result             1 -none-     list
## weights                1 -none-     list
## startweights           1 -none-     list
## generalized.weights    1 -none-     list
## result.matrix         34 -none-     numeric
```

```
plot(nn5, rep="best")
```

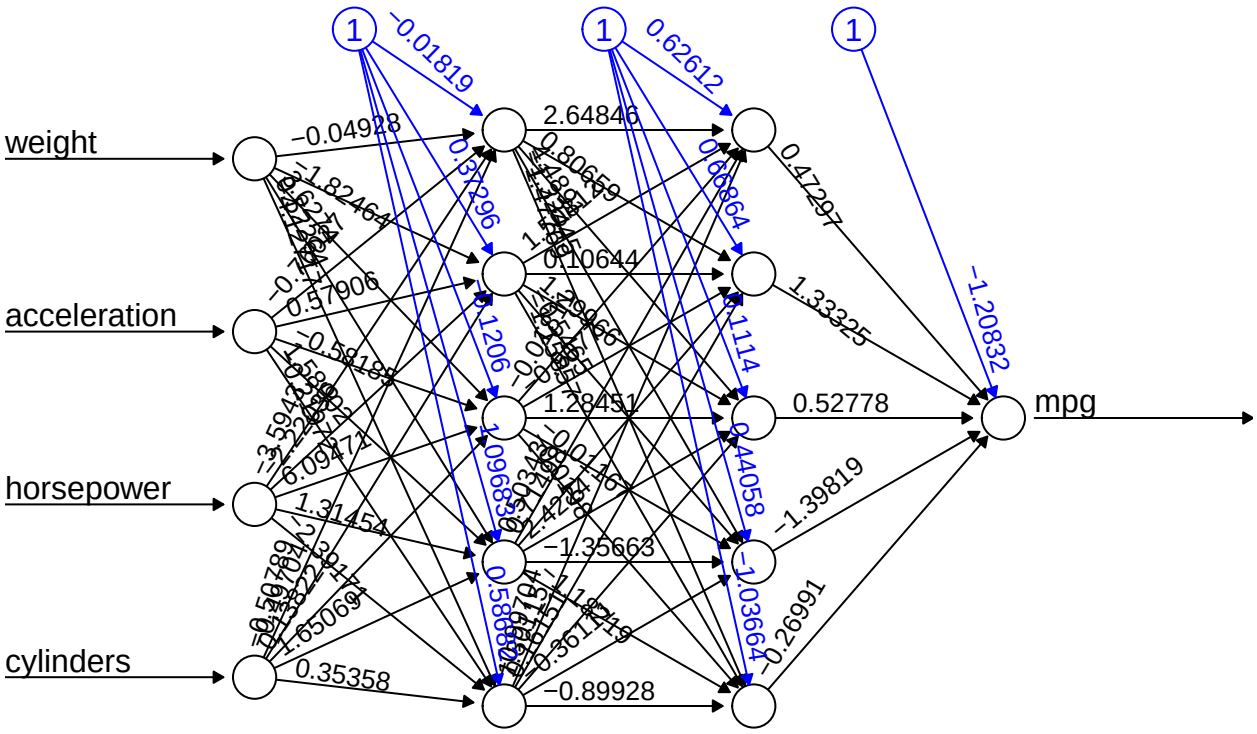



Error: 1.03703  Steps: 404

```
pr.nn5 = compute( nn5,  subset(testData, select=c(weight, acceleration, horsepower, cylinders) ) )

nn55 = neuralnet(mpg ~ weight + acceleration + horsepower + cylinders, data=trainData, hidden=c(5,5))
summary(nn55)
```

```
##                     Length Class      Mode
## call                   4   -none-     call
## response             200   -none-     numeric
## covariate            800   -none-     numeric
## model.list             2   -none-     list
## err.fct                1   -none-     function
## act.fct                1   -none-     function
## linear.output          1   -none-     logical
## data                   9   data.frame list
## net.result             1   -none-     list
## weights                1   -none-     list
## startweights           1   -none-     list
## generalized.weights    1   -none-     list
## result.matrix         64   -none-     numeric
```

```
plot(nn55, rep="best")
```




Error: 1.074624 Steps: 78

```
pr.nn55 = compute( nn55,  subset(testData, select=c(weight, acceleration, horsepower, cylinders) ) )
# Results from NN are normalized (scaled)
# Descaling for comparison
pr.nn0 <- pr.nn0$net.result*(max(Auto$mpg)-min(Auto$mpg))+min(Auto$mpg)
pr.nn5 <- pr.nn5$net.result*(max(Auto$mpg)-min(Auto$mpg))+min(Auto$mpg)
pr.nn55 <- pr.nn55$net.result*(max(Auto$mpg)-min(Auto$mpg))+min(Auto$mpg)
test.r <- (testData$mpg)*(max(Auto$mpg)-min(Auto$mpg))+min(Auto$mpg)

# Calculating MSE
MSE.nn <- sum((test.r - pr.nn0)^2)/nrow(testData)
MSE.nn5 <- sum((test.r - pr.nn5)^2)/nrow(testData)
MSE.nn55 <- sum((test.r - pr.nn55)^2)/nrow(testData)
# Compare the MSEs
print(paste(MSE.lm,MSE.nn,MSE.nn5, MSE.nn55))
```

```
## [1] "17.8324736443234 17.8620306670795 14.5163572372295 15.2882860115222"
```

We see how critical it is to scale our data before building a neural network!!

### Using neural networks to do classification

```
AutoSc$ECO = ifelse( AutoSc$mpg > 0.5, "Economy", "Consuming" )
n = length(AutoSc$mpg)
AutoSc$ECO4 = rep("Economy",n)
AutoSc$ECO4[AutoSc$mpg < 0.75] = "Good"
```

```
AutoSc$ECO4[AutoSc$mpg < 0.5] = "OK"
AutoSc$ECO4[AutoSc$mpg < 0.25] = "Consuming"
trainData <- AutoSc[Z,]
testData <- AutoSc[-Z,]
```

First we will use one output variable. Train an artificial neural network to classify cars into "Economy" and "Consuming".

```
library(nnet)
nn.class = nnet( as.factor(ECO) ~ weight + acceleration + horsepower + cylinders, data=trainData, size=5
```

```
## # weights:  31
## initial  value 136.669815
## iter  10 value 61.420327
## iter  20 value 53.619878
## iter  30 value 52.161995
## iter  40 value 50.549721
## iter  50 value 47.125765
## iter  60 value 45.048199
## iter  70 value 44.533326
## iter  80 value 43.177338
## iter  90 value 40.768818
## iter 100 value 37.656840
## iter 110 value 36.883535
## iter 120 value 35.995247
## iter 130 value 35.623704
## iter 140 value 35.372456
## iter 150 value 35.030882
## iter 160 value 34.723306
## iter 170 value 33.544330
## iter 180 value 32.050584
## iter 190 value 31.212941
## iter 200 value 30.248805
## iter 210 value 29.751090
## iter 220 value 29.277282
## iter 230 value 28.562061
## iter 240 value 28.081954
## iter 250 value 27.582901
## iter 260 value 27.188632
## iter 270 value 27.010850
## iter 280 value 26.899537
## iter 290 value 26.726831
## iter 300 value 26.515571
## iter 310 value 26.163533
## iter 320 value 26.013003
## iter 330 value 25.938772
## iter 340 value 25.785247
## iter 350 value 25.733330
## iter 360 value 25.657508
## iter 370 value 25.448427
## iter 380 value 25.189268
## iter 390 value 25.068564
## iter 400 value 24.823322
## iter 410 value 24.675024
```

```
## iter 420 value 24.573237
## iter 430 value 24.279377
## iter 440 value 23.837118
## iter 450 value 23.776020
## iter 460 value 23.451186
## iter 470 value 23.145696
## iter 480 value 23.017139
## iter 490 value 22.837513
## iter 500 value 22.732903
## final  value 22.732903
## stopped after 500 iterations
```

```
summary(nn.class)
```

```
## a 4-5-1 network with 31 weights
## options were - entropy fitting
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1
##   -1.97 -241.81  110.44 -104.02  107.53
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2
##  -49.78   -6.22   43.05  102.38   20.13
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3
##   16.12  -73.86  -18.29  -26.72   13.78
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4
##  -84.48    9.81   73.13  163.75   22.20
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5
##   60.38 -191.88  -52.98 -117.51  -12.11
##    b->o   h1->o   h2->o   h3->o   h4->o   h5->o
##   -3.12    4.81  -77.67  133.58   74.30  -92.33
```

```
pr.nn = predict(nn.class, subset(testData, select=c(weight,acceleration,horsepower,cylinders) ), type =
head(pr.nn)
```

```
## [1] "Consuming" "Consuming" "Consuming" "Consuming" "Consuming" "Consuming"
```

```
table(pr.nn,testData$ECO)
```

```
##
## pr.nn       Consuming Economy
##   Consuming       112      18
##   Economy          21      41
```

```
# train neural network on 4 inputs, 1 hidden layer with 5 nodes, 4 output nodes
nn4.class = nnet( as.factor(ECO4) ~ weight+acceleration+horsepower+cylinders, data=trainData, size=5 , r
```

```
## # weights:  49
## initial  value 285.788088
## iter  10 value 147.621204
## iter  20 value 127.164456
## iter  30 value 119.910773
## iter  40 value 110.919763
## iter  50 value 105.414131
## iter  60 value 102.323318
## iter  70 value 100.340219
## iter  80 value 98.912595
## iter  90 value 97.952976
## iter 100 value 96.803550
## iter 110 value 95.867543
```

```
## iter 120 value 94.644272
## iter 130 value 93.560171
## iter 140 value 92.840335
## iter 150 value 92.468186
## iter 160 value 92.260835
## iter 170 value 92.137376
## iter 180 value 91.895166
## iter 190 value 91.740578
## iter 200 value 91.584289
## iter 210 value 91.567647
## iter 220 value 91.521571
## iter 230 value 91.485273
## iter 240 value 91.424759
## iter 250 value 91.395310
## iter 260 value 91.364708
## iter 270 value 91.351189
## iter 280 value 91.309009
## iter 290 value 91.232543
## iter 300 value 91.183813
## iter 310 value 91.175779
## iter 320 value 91.156833
## iter 330 value 91.146345
## iter 340 value 91.136255
## iter 350 value 91.133364
## iter 360 value 91.129454
## iter 370 value 91.123012
## iter 380 value 91.110374
## iter 390 value 91.097234
## iter 400 value 91.092726
## iter 410 value 91.092145
## iter 420 value 91.079176
## iter 430 value 91.073899
## iter 440 value 91.071119
## iter 450 value 91.069591
## iter 460 value 91.068497
## iter 470 value 91.063148
## iter 480 value 91.060869
## iter 490 value 91.056823
## iter 500 value 91.027115
## final  value 91.027115
## stopped after 500 iterations
```

```
summary(nn4.class)
```

```
## a 4-5-4 network with 49 weights
## options were - softmax modelling
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1
##   -3.71   13.80    1.74   -2.04   -3.85
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2
##    3.43  -14.59   -1.89    6.41    2.05
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3
##   -2.97    8.59    0.63    3.87   -2.94
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4
##   75.79   88.60 -129.59   -8.30   24.73
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5
```

```
##  -17.67    5.47    1.33   -8.25   30.30
##   b->o1  h1->o1  h2->o1  h3->o1  h4->o1  h5->o1
##  -32.21    8.14   24.21   28.65    1.47   19.96
##   b->o2  h1->o2  h2->o2  h3->o2  h4->o2  h5->o2
##   70.45  -74.09  -73.26  -24.71    5.49    8.58
##   b->o3  h1->o3  h2->o3  h3->o3  h4->o3  h5->o3
##  -53.75   97.66   61.93  -38.24   -1.10  -37.17
##   b->o4  h1->o4  h2->o4  h3->o4  h4->o4  h5->o4
##   15.84  -32.93  -13.61   35.39   -5.85    9.39
```

```
names(nn4.class)
```

```
##  [1] "n"             "nunits"        "nconn"         "conn"
##  [5] "nsunits"       "decay"         "entropy"       "softmax"
##  [9] "censored"      "value"         "wts"           "convergence"
## [13] "fitted.values" "residuals"     "lev"           "call"
## [17] "terms"         "coefnames"     "xlevels"
```

```
pr.nn4 = predict(nn4.class, subset(testData, select=c(weight,acceleration,horsepower,cylinders) ), type
head(pr.nn4)
```

```
## [1] "OK"        "Consuming" "Consuming" "Consuming" "Consuming" "Consuming"
```

```
table(pr.nn4,testData$ECO4)
```

```
##
## pr.nn4      Consuming Economy Good OK
##   Consuming        53       0    0  9
##   Economy           0       3    3  3
##   Good              0       6   26 16
##   OK                9       0   21 43
```