Neural Net Package Examples

Raphael Cobe

12/2018

```
library("neuralnet")
```

Going to create a neural network to perform square rooting Type ?neuralnet for more information on the neuralnet library

Generate 50 random numbers uniformly distributed between 0 and 100 And store them as a dataframe

```
traininginput <-  as.data.frame(runif(50, min=0, max=100))
trainingoutput <- sqrt(traininginput)
```

Column bind the data into one variable

```
trainingdata <- cbind(traininginput,trainingoutput)
colnames(trainingdata) <- c("Input","Output")
```

Train the neural network Going to have 10 hidden layers Threshold is a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.

```
net.sqrt <- neuralnet(Output~Input,trainingdata, hidden=10, threshold=0.01)
print(net.sqrt)
```

```
## $call
## neuralnet(formula = Output ~ Input, data = trainingdata, hidden = 10,
##     threshold = 0.01)
##
## $response
##          Output
## 1  9.6217808694
## 2  2.7542597843
## 3  7.2048247263
## 4  7.6506244616
## 5  4.7660272289
## 6  8.7080475438
## 7  0.4531911471
## 8  7.1840535604
## 9  9.8020647574
## 10 8.4477154923
## 11 9.8571039534
## 12 8.6388046801
## 13 9.3866483264
## 14 4.9045332117
## 15 7.6031568998
## 16 7.7931112053
## 17 7.2222735027
## 18 4.8656745153
## 19 6.7490742636
## 20 9.2461595138
## 21 4.4496398598
## 22 6.8940508148
## 23 7.3701520504
```

```
## 24 7.8754306286
## 25 8.2935691230
## 26 6.3437736668
## 27 9.3145856575
## 28 2.8642456703
## 29 5.0305061785
## 30 8.9011377273
## 31 8.8377206112
## 32 9.9626666758
## 33 2.7143503803
## 34 6.0110522902
## 35 3.8565914310
## 36 5.1387666633
## 37 9.0869065256
## 38 6.1745455271
## 39 6.7364707118
## 40 5.8430417600
## 41 8.0964897574
## 42 4.7642399434
## 43 5.9402377043
## 44 7.9201128530
## 45 5.3466353785
## 46 7.4399063975
## 47 6.6623556583
## 48 5.0860434152
## 49 7.6154663666
## 50 8.4605759607
##
## $covariate
##                [,1]
##  [1,] 92.5786670996
##  [2,]  7.5859469594
##  [3,] 51.9094993360
##  [4,] 58.5320546525
##  [5,] 22.7150155464
##  [6,] 75.8300920250
##  [7,]  0.2053822158
##  [8,] 51.6106255585
##  [9,] 96.0804735078
## [10,] 71.3638970396
## [11,] 97.1624983475
## [12,] 74.6289463015
## [13,] 88.1091668038
## [14,] 24.0544460248
## [15,] 57.8079948435
## [16,] 60.7325822581
## [17,] 52.1612345474
## [18,] 23.6747884890
## [19,] 45.5500034150
## [20,] 85.4914657539
## [21,] 19.7992948815
## [22,] 47.5279366365
## [23,] 54.3191412464
## [24,] 62.0224075858
```

```
## [25,] 68.7832887983
## [26,] 40.2434643358
## [27,] 86.7615059717
## [28,]  8.2039032597
## [29,] 25.3059924114
## [30,] 79.2302528396
## [31,] 78.1053056009
## [32,] 99.2547272937
## [33,]  7.3676979868
## [34,] 36.1327496357
## [35,] 14.8732974660
## [36,] 26.4069228200
## [37,] 82.5718702050
## [38,] 38.1250124658
## [39,] 45.3800376505
## [40,] 34.1411370086
## [41,] 65.5531463912
## [42,] 22.6979822386
## [43,] 35.2864239831
## [44,] 62.7281876048
## [45,] 28.5865098704
## [46,] 55.3522072034
## [47,] 44.3869829178
## [48,] 25.8678376209
## [49,] 57.9953279812
## [50,] 71.5813455870
##
## $model.list
## $model.list$response
## [1] "Output"
##
## $model.list$variables
## [1] "Input"
##
##
## $err.fct
## function (x, y)
## {
##     1/2 * (y - x)^2
## }
## <bytecode: 0x7fb99d02ca38>
## <environment: 0x7fb99d02bb20>
## attr(,"type")
## [1] "sse"
##
## $act.fct
## function (x)
## {
##     1/(1 + exp(-x))
## }
## <bytecode: 0x7fb99c8ff300>
## <environment: 0x7fb99d02bb20>
## attr(,"type")
## [1] "logistic"
```

```
##
## $linear.output
## [1] TRUE
##
## $data
##            Input       Output
## 1  92.5786670996 9.6217808694
## 2   7.5859469594 2.7542597843
## 3  51.9094993360 7.2048247263
## 4  58.5320546525 7.6506244616
## 5  22.7150155464 4.7660272289
## 6  75.8300920250 8.7080475438
## 7   0.2053822158 0.4531911471
## 8  51.6106255585 7.1840535604
## 9  96.0804735078 9.8020647574
## 10 71.3638970396 8.4477154923
## 11 97.1624983475 9.8571039534
## 12 74.6289463015 8.6388046801
## 13 88.1091668038 9.3866483264
## 14 24.0544460248 4.9045332117
## 15 57.8079948435 7.6031568998
## 16 60.7325822581 7.7931112053
## 17 52.1612345474 7.2222735027
## 18 23.6747884890 4.8656745153
## 19 45.5500034150 6.7490742636
## 20 85.4914657539 9.2461595138
## 21 19.7992948815 4.4496398598
## 22 47.5279366365 6.8940508148
## 23 54.3191412464 7.3701520504
## 24 62.0224075858 7.8754306286
## 25 68.7832887983 8.2935691230
## 26 40.2434643358 6.3437736668
## 27 86.7615059717 9.3145856575
## 28  8.2039032597 2.8642456703
## 29 25.3059924114 5.0305061785
## 30 79.2302528396 8.9011377273
## 31 78.1053056009 8.8377206112
## 32 99.2547272937 9.9626666758
## 33  7.3676979868 2.7143503803
## 34 36.1327496357 6.0110522902
## 35 14.8732974660 3.8565914310
## 36 26.4069228200 5.1387666633
## 37 82.5718702050 9.0869065256
## 38 38.1250124658 6.1745455271
## 39 45.3800376505 6.7364707118
## 40 34.1411370086 5.8430417600
## 41 65.5531463912 8.0964897574
## 42 22.6979822386 4.7642399434
## 43 35.2864239831 5.9402377043
## 44 62.7281876048 7.9201128530
## 45 28.5865098704 5.3466353785
## 46 55.3522072034 7.4399063975
## 47 44.3869829178 6.6623556583
## 48 25.8678376209 5.0860434152
```

```
## 49 57.9953279812 7.6154663666
## 50 71.5813455870 8.4605759607
##
## $net.result
## $net.result[[1]]
##             [,1]
## 1  9.6217816379
## 2  2.7552929279
## 3  7.2005673011
## 4  7.6456017167
## 5  4.7646677217
## 6  8.7153060190
## 7  0.4531622752
## 8  7.1799368459
## 9  9.7935239050
## 10 8.4514444643
## 11 9.8452860801
## 12 8.6452086047
## 13 9.3933674681
## 14 4.9025441111
## 15 7.5979923296
## 16 7.7888376683
## 17 7.2179040776
## 18 4.8637972645
## 19 6.7491613356
## 20 9.2549063656
## 21 4.4516081766
## 22 6.8925623226
## 23 7.3650813808
## 24 7.8718008745
## 25 8.2950290338
## 26 6.3476598441
## 27 9.3225097471
## 28 2.8563825681
## 29 5.0285039613
## 30 8.9101965811
## 31 8.8463022034
## 32 9.9436101376
## 33 2.7196564885
## 34 6.0158224285
## 35 3.8603710082
## 36 5.1371510730
## 37 9.0965034676
## 38 6.1792153291
## 39 6.7366959433
## 40 5.8472240834
## 41 8.0952451384
## 42 4.7628926536
## 43 5.9448424552
## 44 7.9168925341
## 45 5.3465416983
## 46 7.4346720672
## 47 6.6633844699
## 48 5.0841970660
```

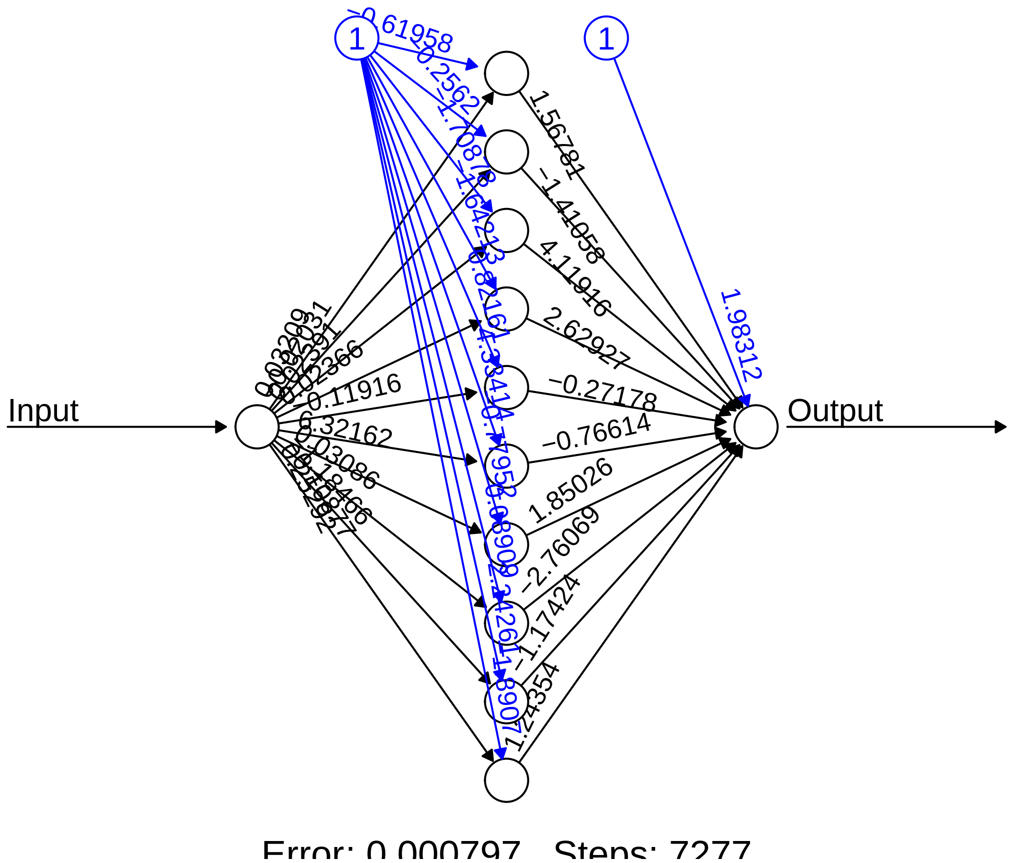
```
## 49 7.6103333132
## 50 8.4644935238
##
##
## $weights
## $weights[[1]]
## $weights[[1]][[1]]
##                [,1]          [,2]          [,3]         [,4]
## [1,] -0.61957621275 -0.25620273250 -1.70877761552 -1.6421308619
## [2,]  0.03208577082 -0.02031377771  0.02390804943  0.0236621543
##               [,5]        [,6]           [,7]           [,8]          [,9]
## [1,]  0.8216118714 4.334135880 -0.77951521955  0.08908873886  2.2426085181
## [2,] -0.1191573148 6.321616384  0.03086454671 -1.18466421757 -0.1087727731
##              [,10]
## [1,] -1.8907034024
## [2,]  0.2529188421
##
## $weights[[1]][[2]]
##                [,1]
##  [1,]  1.9831229849
##  [2,]  1.5678124783
##  [3,] -1.4105774377
##  [4,]  4.1191574635
##  [5,]  2.6292737573
##  [6,] -0.2717785769
##  [7,] -0.7661444095
##  [8,]  1.8502566826
##  [9,] -2.7606947764
## [10,] -1.1742379883
## [11,]  1.2435420046
##
##
##
## $startweights
## $startweights[[1]]
## $startweights[[1]][[1]]
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,] -0.9533468516 -1.0302995075 -1.0856777800 -1.5351913291 -1.2777707570
## [2,] -0.1966290665 -0.7674577627  0.2570134446 -0.2644647865 -0.6379691354
##              [,6]           [,7]          [,8]         [,9]        [,10]
## [1,] -0.466919533 -0.70512847089 -0.3634019617  2.117401420 -1.362224106
## [2,]  1.520560972 -0.03333466286 -1.4357863286 -1.150343607  2.352212947
##
## $startweights[[1]][[2]]
##                [,1]
##  [1,]  1.2485872879
##  [2,]  0.6445258725
##  [3,] -0.6332755781
##  [4,]  1.3183326930
##  [5,]  0.9239325901
##  [6,]  0.0381179118
##  [7,] -1.5006801065
##  [8,]  0.9256667598
##  [9,]  1.8642395482
```

```
## [10,] -0.6165355383
## [11,]  0.5101595872
##
##
##
## $generalized.weights
## $generalized.weights[[1]]
##                 [,1]
## 1  -0.0006022120832
## 2  -0.0337873911931
## 3  -0.0015441165053
## 4  -0.0012908391791
## 5  -0.0058088139361
## 6  -0.0008639047466
## 7   3.6446807560163
## 8  -0.0015576181184
## 9  -0.0005588097533
## 10 -0.0009536557311
## 11 -0.0005460403631
## 12 -0.0008870121047
## 13 -0.0006626009973
## 14 -0.0053166612456
## 15 -0.0013149552111
## 16 -0.0012217523198
## 17 -0.0015329102005
## 18 -0.0054487599924
## 19 -0.0018883434523
## 20 -0.0007008381817
## 21 -0.0072213991230
## 22 -0.0017667984338
## 23 -0.0014425915240
## 24 -0.0011839002318
## 25 -0.0010107669119
## 26 -0.0023068738510
## 27 -0.0006820065902
## 28 -0.0308593388328
## 29 -0.0049161327345
## 30 -0.0008021622544
## 31 -0.0008220174711
## 32 -0.0005221613537
## 33 -0.0348832548876
## 34 -0.0027606596941
## 35 -0.0119129721071
## 36 -0.0046014329132
## 37 -0.0007462478785
## 38 -0.0025235421790
## 39 -0.0018995289000
## 40 -0.0030356062650
## 41 -0.0010887343243
## 42 -0.0058155752825
## 43 -0.0028725244791
## 44 -0.0011639329681
## 45 -0.0040604547923
## 46 -0.0014026440298
```

```
## 47 -0.0019674602291
## 48 -0.0047515712853
## 49 -0.0013086452412
## 50 -0.0009490314384
##
##
## $result.matrix
##                                          1
## error                      0.0007970288253
## reached.threshold          0.0084048487287
## steps                   7277.0000000000000
## Intercept.to.1layhid1     -0.6195762127474
## Input.to.1layhid1          0.0320857708230
## Intercept.to.1layhid2     -0.2562027324961
## Input.to.1layhid2         -0.0203137777137
## Intercept.to.1layhid3     -1.7087776155228
## Input.to.1layhid3          0.0239080494312
## Intercept.to.1layhid4     -1.6421308618898
## Input.to.1layhid4          0.0236621542992
## Intercept.to.1layhid5      0.8216118714349
## Input.to.1layhid5         -0.1191573147635
## Intercept.to.1layhid6      4.3341358795528
## Input.to.1layhid6          6.3216163844058
## Intercept.to.1layhid7     -0.7795152195484
## Input.to.1layhid7          0.0308645467109
## Intercept.to.1layhid8      0.0890887388579
## Input.to.1layhid8         -1.1846642175729
## Intercept.to.1layhid9      2.2426085180761
## Input.to.1layhid9         -0.1087727731109
## Intercept.to.1layhid10    -1.8907034023895
## Input.to.1layhid10         0.2529188420863
## Intercept.to.Output        1.9831229849260
## 1layhid.1.to.Output        1.5678124783108
## 1layhid.2.to.Output       -1.4105774376918
## 1layhid.3.to.Output        4.1191574634507
## 1layhid.4.to.Output        2.6292737572629
## 1layhid.5.to.Output       -0.2717785768667
## 1layhid.6.to.Output       -0.7661444095427
## 1layhid.7.to.Output        1.8502566825762
## 1layhid.8.to.Output       -2.7606947764139
## 1layhid.9.to.Output       -1.1742379882711
## 1layhid.10.to.Output       1.2435420046325
##
## attr(,"class")
## [1] "nn"
```

Plot the neural network

```r
plot(net.sqrt, rep = "best")
```



Test the neural network on some training data

```
testdata <- as.data.frame((1:10)^2) #Generate some squared numbers
net.results <- compute(net.sqrt, testdata) #Run them through the neural network
```

Lets see what properties net.sqrt has

```
ls(net.results)
```

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

Lets see the results

```
print(net.results$net.result)
```

```
##              [,1]
##  [1,] 1.121132769
##  [2,] 2.173361935
##  [3,] 2.986371642
##  [4,] 4.005102928
##  [5,] 4.997952969
##  [6,] 6.004752544
##  [7,] 6.997433685
##  [8,] 7.997608343
##  [9,] 9.009522588
## [10,] 9.978069512
```

Lets display a better version of the results

```r
cleanoutput <- cbind(testdata,sqrt(testdata),
                     as.data.frame(net.results$net.result))
colnames(cleanoutput) <- c("Input","Expected Output","Neural Net Output")
print(cleanoutput)
```

```
##    Input Expected Output Neural Net Output
## 1      1               1       1.121132769
## 2      4               2       2.173361935
## 3      9               3       2.986371642
## 4     16               4       4.005102928
## 5     25               5       4.997952969
## 6     36               6       6.004752544
## 7     49               7       6.997433685
## 8     64               8       7.997608343
## 9     81               9       9.009522588
## 10   100              10       9.978069512
```

sin function

Generate random data and the dependent variable

```r
x <- sort(runif(50, min = 0, max = 4*pi))
y <- sin(x)

data <- cbind(x,y)
```

Create the neural network responsible for the sin function

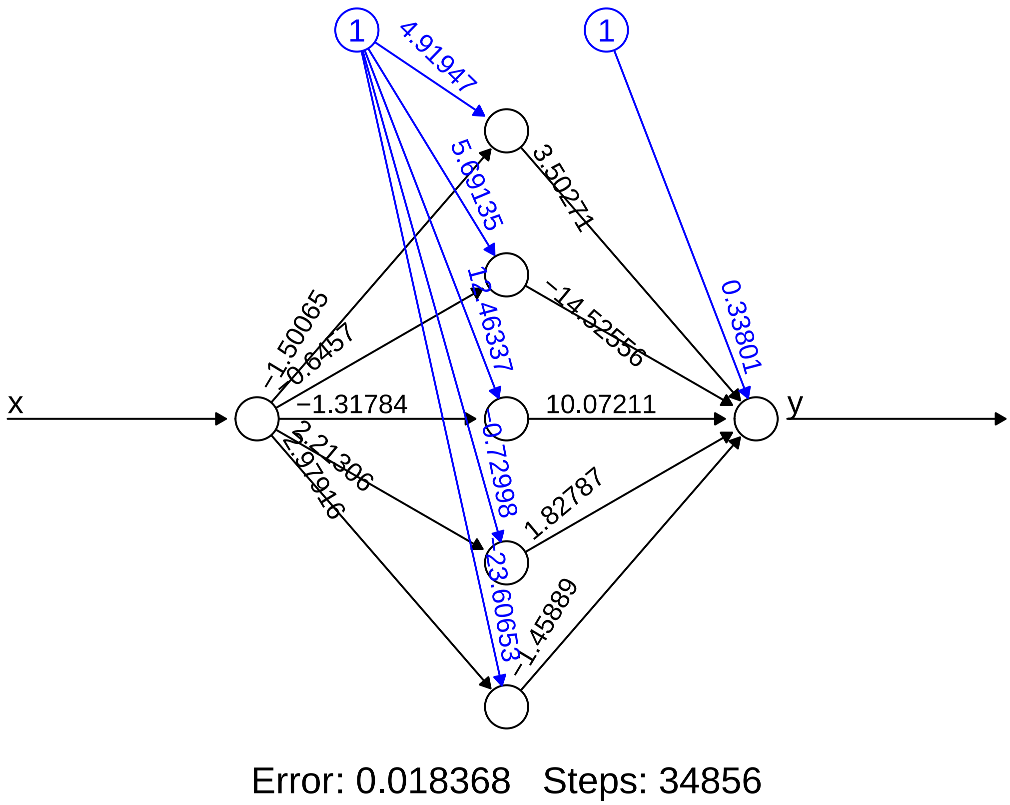
```r
library(neuralnet)
sin.nn <- neuralnet(y ~ x, data = data, hidden = 5, stepmax = 100000, learningrate = 10e-6,
                    act.fct = 'logistic', err.fct = 'sse', rep = 5, lifesign = "minimal",
                    linear.output = T)
```

```
## hidden: 5    thresh: 0.01    rep: 1/5    steps:   26224  error: 0.04795  time: 5.52 secs
## hidden: 5    thresh: 0.01    rep: 2/5    steps: stepmax  min thresh: 0.01099820668
## hidden: 5    thresh: 0.01    rep: 3/5    steps:   89581  error: 0.0424   time: 12.19 secs
## hidden: 5    thresh: 0.01    rep: 4/5    steps:   34856  error: 0.01837  time: 4.87 secs
## hidden: 5    thresh: 0.01    rep: 5/5    steps:   58988  error: 0.03869  time: 8.7 secs

## Warning: algorithm did not converge in 1 of 5 repetition(s) within the
## stepmax
```

Visualize the neural network

```r
plot(sin.nn, rep = "best")
```



Generate data for the prediction of the using the neural net;

```
testdata<- as.data.frame(runif(10, min=0, max=(4*pi)))
testdata
```

```
##    runif(10, min = 0, max = (4 * pi))
## 1                         0.936245349
## 2                        11.855929155
## 3                         6.924013830
## 4                         7.035336691
## 5                         8.197465744
## 6                        10.736771671
## 7                         2.574070558
## 8                        10.548511453
## 9                         8.412578490
## 10                        8.342025594
```

Calculate the real value using the `sin` function

```
testdata.result <- sin(testdata)
```

Make the prediction

```
sin.nn.result <- compute(sin.nn, testdata)
sin.nn.result$net.result
```

```
##                [,1]
##  [1,]  0.7624888424
##  [2,] -0.6818526591
##  [3,]  0.4843483634
##  [4,]  0.5658853677
```

```
##  [5,]  0.9636076126
##  [6,] -0.9479874888
##  [7,]  0.5338347014
##  [8,] -0.9121576256
##  [9,]  0.8981825602
## [10,]  0.9255634163
```

Compare with the real values:

```
better <- cbind(testdata, sin.nn.result$net.result, testdata.result, (sin.nn.result$net.result-testdata
colnames(better) <- c("Input", "NN Result", "Result", "Error")

better
```

```
##           Input     NN Result        Result           Error
## 1   0.936245349  0.7624888424  0.8053379652 -0.042849122808
## 2  11.855929155 -0.6818526591 -0.6521684935 -0.029684165613
## 3   6.924013830  0.4843483634  0.5978597912 -0.113511427834
## 4   7.035336691  0.5658853677  0.6832113250 -0.117325957239
## 5   8.197465744  0.9636076126  0.9415870403  0.022020572370
## 6  10.736771671 -0.9479874888 -0.9666971093  0.018709620471
## 7   2.574070558  0.5338347014  0.5375442445 -0.003709543108
## 8  10.548511453 -0.9121576256 -0.9017207820 -0.010436843614
## 9   8.412578490  0.8981825602  0.8479996073  0.050182952851
## 10  8.342025594  0.9255634163  0.8832517331  0.042311683128
```

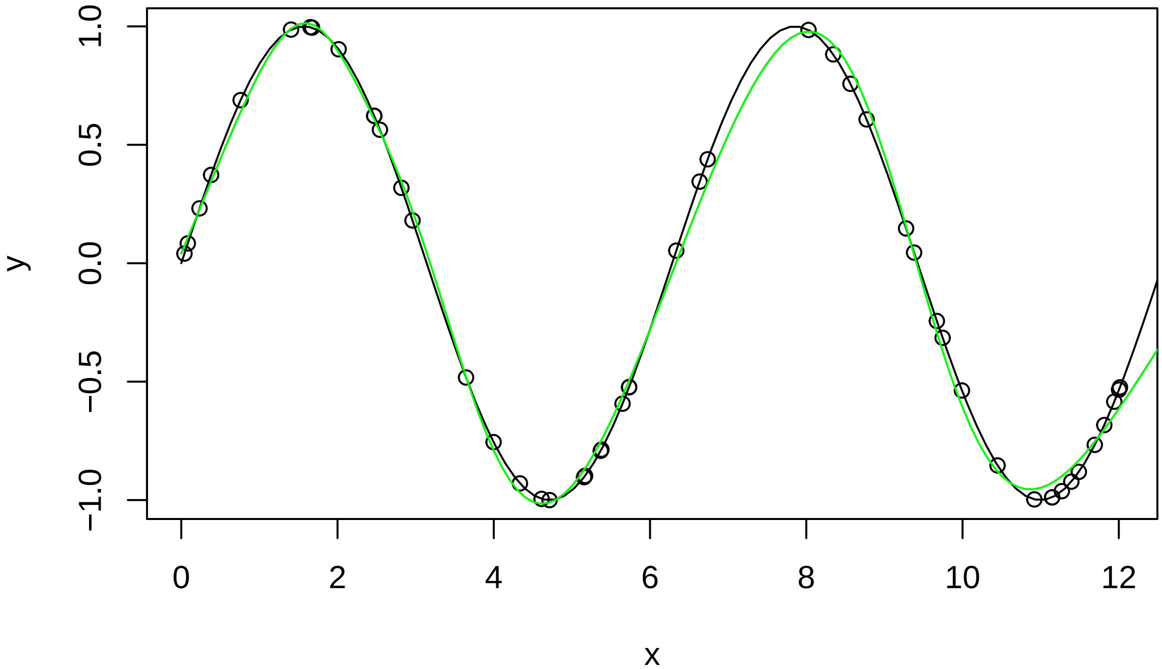
Calculate the RMSE:

```
library(Metrics)
rmse(better$Result, better$`NN Result`)
```

```
## [1] 0.05885038038
```

Plot the results:

```
plot(x,y)
plot(sin, 0, (4*pi), add=T)
x1 <- seq(0, 4*pi, by=0.1)
lines(x1, compute(sin.nn, data.frame(x=x1))$net.result, col="green")
```



A classification problem

Using the `iris` dataset

```
data(iris)
iris.dataset <- iris
```

Check what is inside the dataset:

```
head(iris.dataset)
```

```
##   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
```

Change the dataset so we are able to predict classes:

```
iris.dataset$setosa <- iris.dataset$Species=="setosa"
iris.dataset$virginica = iris.dataset$Species == "virginica"
iris.dataset$versicolor = iris.dataset$Species == "versicolor"
```

Separate into train and test data:

```
train <- sample(x = nrow(iris.dataset), size = nrow(iris)*0.5)
train
```

```
##  [1] 148  28  19  53  85  18 136  83  34  36 122 133  44   5 119  43  20
## [18]  63   1  72 115 127  92  15  42 106 118 128  73   9 113  11 100  99
## [35] 120 117 111 131 116  48   3 114   6  39 130  29 107  65 123   7 124
## [52]  96  68  27 138  46  33  84  37 102  98 134  80  49  58  32 112  91
```

```
## [69]  14  47 104 137 125 150  60
```

```
iristrain <- iris.dataset[train,]
irisvalid <- iris.dataset[-train,]
print(nrow(iristrain))
```

```
## [1] 75
```

```
print(nrow(irisvalid))
```

```
## [1] 75
```

Build the Neural Network for the classification:

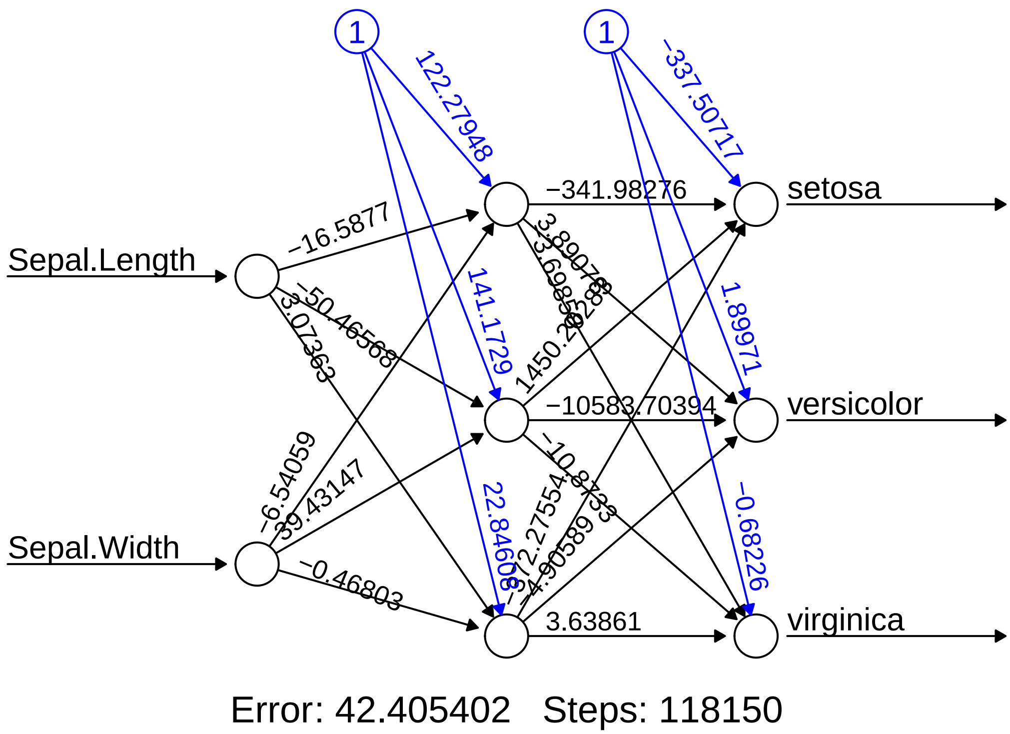
```
nn <- neuralnet(setosa+versicolor+virginica ~ Sepal.Length + Sepal.Width, data=iristrain, hidden=3,
                rep = 2, err.fct = "ce", linear.output = F, lifesign = "minimal", stepmax = 1000000)
```

```
## hidden: 3    thresh: 0.01    rep: 1/2    steps: stepmax  min thresh: 0.03576456698
## hidden: 3    thresh: 0.01    rep: 2/2    steps:  118150  error: 42.4054  time: 18.57 secs
```

```
## Warning: algorithm did not converge in 1 of 2 repetition(s) within the
## stepmax
```

Let's check the neural network that we just built

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



Let's try to make the prediction:

```
comp <- compute(nn, irisvalid[-3:-8])
pred.weights <- comp$net.result
idx <- apply(pred.weights, 1, which.max)
pred <- c('setosa', 'versicolor', 'virginica')[idx]
table(pred, irisvalid$Species)
```

```
##
```

```
## pred         setosa versicolor virginica
##   setosa         22          0         0
##   versicolor      0         20         4
##   virginica       0         12        17
```

AND operation

```r
AND <- c(rep(0,3),1)
OR <- c(0,rep(1,3))
binary.data <- data.frame(expand.grid(c(0,1), c(0,1)), AND)
print(net <- neuralnet(AND~Var1+Var2, binary.data, hidden=0, rep=10, err.fct="ce", linear.output=FALSE))
```

```
## $call
## neuralnet(formula = AND ~ Var1 + Var2, data = binary.data, hidden = 0,
##     rep = 10, err.fct = "ce", linear.output = FALSE)
##
## $response
##   AND
## 1   0
## 2   0
## 3   0
## 4   1
##
## $covariate
##      [,1] [,2]
## [1,]    0    0
## [2,]    1    0
## [3,]    0    1
## [4,]    1    1
##
## $model.list
## $model.list$response
## [1] "AND"
##
## $model.list$variables
## [1] "Var1" "Var2"
##
##
## $err.fct
## function (x, y)
## {
##     -(y * log(x) + (1 - y) * log(1 - x))
## }
## <bytecode: 0x7fb99c8fe928>
## <environment: 0x7fb99da0c338>
## attr(,"type")
## [1] "ce"
##
## $act.fct
## function (x)
## {
##     1/(1 + exp(-x))
```

```
## }
## <bytecode: 0x7fb99c8ff300>
## <environment: 0x7fb99da0c338>
## attr(,"type")
## [1] "logistic"
##
## $linear.output
## [1] FALSE
##
## $data
##   Var1 Var2 AND
## 1    0    0   0
## 2    1    0   0
## 3    0    1   0
## 4    1    1   1
##
## $net.result
## $net.result[[1]]
##                 [,1]
## 1 0.000004106937754
## 2 0.014115001822988
## 3 0.014694032186124
## 4 0.981127868153618
##
## $net.result[[2]]
##                 [,1]
## 1 0.000004614482444
## 2 0.014508528295724
## 3 0.013618778386834
## 4 0.977802088539118
##
## $net.result[[3]]
##                 [,1]
## 1 0.000003938758384
## 2 0.014364012149706
## 3 0.013526700382011
## 4 0.980670596237003
##
## $net.result[[4]]
##                 [,1]
## 1 0.000006433909379
## 2 0.015306884982742
## 3 0.016362710268807
## 4 0.975722831767955
##
## $net.result[[5]]
##                 [,1]
## 1 0.000008824801607
## 2 0.017724080040923
## 3 0.017557735099988
## 4 0.973362603203442
##
## $net.result[[6]]
##                 [,1]
```

```
## 1 0.000003886227609
## 2 0.014006516475723
## 3 0.014072792527809
## 4 0.981194133950949
##
## $net.result[[7]]
##                 [,1]
## 1 0.000009561827169
## 2 0.018558273364605
## 3 0.017649149297976
## 4 0.972624623441297
##
## $net.result[[8]]
##                 [,1]
## 1 0.000004626434606
## 2 0.014513855321458
## 3 0.015198419678493
## 4 0.980051269191773
##
## $net.result[[9]]
##                 [,1]
## 1 0.000001864345151
## 2 0.010952238085935
## 3 0.011132093265940
## 4 0.985264803544051
##
## $net.result[[10]]
##                 [,1]
## 1 0.000002797194923
## 2 0.012820665331455
## 3 0.011832221399597
## 4 0.982330211944017
##
##
## $weights
## $weights[[1]]
## $weights[[1]][[1]]
##               [,1]
## [1,] -12.40282877
## [2,]   8.15652725
## [3,]   8.19731799
##
##
## $weights[[2]]
## $weights[[2]][[1]]
##                [,1]
## [1,] -12.286306228
## [2,]   8.067902390
## [3,]   8.003712918
##
##
## $weights[[3]]
## $weights[[3]][[1]]
##                [,1]
```

```
## [1,] -12.444641077
## [2,]   8.216079893
## [3,]   8.155170355
##
##
## $weights[[4]]
## $weights[[4]][[1]]
##               [,1]
## [1,] -11.953921780
## [2,]   7.789894472
## [3,]   7.857669541
##
##
## $weights[[5]]
## $weights[[5]][[1]]
##               [,1]
## [1,] -11.637935611
## [2,]   7.622987534
## [3,]   7.613388632
##
##
## $weights[[6]]
## $weights[[6]][[1]]
##               [,1]
## [1,] -12.458067751
## [2,]   8.203940689
## [3,]   8.208728551
##
##
## $weights[[7]]
## $weights[[7]][[1]]
##               [,1]
## [1,] -11.557722161
## [2,]   7.589615214
## [3,]   7.538461218
##
##
## $weights[[8]]
## $weights[[8]][[1]]
##               [,1]
## [1,] -12.283719423
## [2,]   8.065688089
## [3,]   8.112470698
##
##
## $weights[[9]]
## $weights[[9]][[1]]
##               [,1]
## [1,] -13.192598828
## [2,]   8.689400031
## [3,]   8.705870289
##
##
## $weights[[10]]
```

```
## $weights[[10]][[1]]
##              [,1]
## [1,] -12.786890659
## [2,]   8.443097287
## [3,]   8.361864596
##
##
##
## $startweights
## $startweights[[1]]
## $startweights[[1]][[1]]
##               [,1]
## [1,] -1.43442877212
## [2,] -0.09587275012
## [3,] -0.51828201042
##
##
## $startweights[[2]]
## $startweights[[2]][[1]]
##              [,1]
## [1,] -0.1863062282
## [2,] -0.1844976097
## [3,] -0.5802870821
##
##
## $startweights[[3]]
## $startweights[[3]][[1]]
##              [,1]
## [1,]  0.9553589234
## [2,] -0.4731201070
## [3,] -0.0781067645
##
##
## $startweights[[4]]
## $startweights[[4]][[1]]
##              [,1]
## [1,]  0.5460782200
## [2,] -0.3993055282
## [3,] -0.6631304593
##
##
## $startweights[[5]]
## $startweights[[5]][[1]]
##              [,1]
## [1,]  0.4620643887
## [2,] -0.6294124658
## [3,] -0.9706113680
##
##
## $startweights[[6]]
## $startweights[[6]][[1]]
##              [,1]
## [1,]  0.7419322490
## [2,] -1.3484593107
```

```
## [3,]  0.3754514306
##
##
## $startweights[[7]]
## $startweights[[7]][[1]]
##               [,1]
## [1,] 1.44227783921
## [2,] 0.50041521368
## [3,] 0.04926121827
##
##
## $startweights[[8]]
## $startweights[[8]][[1]]
##                [,1]
## [1,]  0.51628057651
## [2,] -0.09191191122
## [3,] -1.83472930216
##
##
## $startweights[[9]]
## $startweights[[9]][[1]]
##               [,1]
## [1,] -0.6925988277
## [2,] -0.4945999691
## [3,]  0.7166702892
##
##
## $startweights[[10]]
## $startweights[[10]][[1]]
##               [,1]
## [1,]  0.1131093410
## [2,] -0.7409027126
## [3,]  1.9239874764
##
##
##
## $generalized.weights
## $generalized.weights[[1]]
##            [,1]       [,2]
## [1,] 8.15652725 8.19731799
## [2,] 8.15652725 8.19731799
## [3,] 8.15652725 8.19731799
## [4,] 8.15652725 8.19731799
##
## $generalized.weights[[2]]
##            [,1]        [,2]
## [1,] 8.06790239 8.003712918
## [2,] 8.06790239 8.003712918
## [3,] 8.06790239 8.003712918
## [4,] 8.06790239 8.003712918
##
## $generalized.weights[[3]]
##             [,1]        [,2]
## [1,] 8.216079893 8.155170355
```

```
## [2,] 8.216079893 8.155170355
## [3,] 8.216079893 8.155170355
## [4,] 8.216079893 8.155170355
##
## $generalized.weights[[4]]
##             [,1]        [,2]
## [1,] 7.789894472 7.857669541
## [2,] 7.789894472 7.857669541
## [3,] 7.789894472 7.857669541
## [4,] 7.789894472 7.857669541
##
## $generalized.weights[[5]]
##             [,1]        [,2]
## [1,] 7.622987534 7.613388632
## [2,] 7.622987534 7.613388632
## [3,] 7.622987534 7.613388632
## [4,] 7.622987534 7.613388632
##
## $generalized.weights[[6]]
##             [,1]        [,2]
## [1,] 8.203940689 8.208728551
## [2,] 8.203940689 8.208728551
## [3,] 8.203940689 8.208728551
## [4,] 8.203940689 8.208728551
##
## $generalized.weights[[7]]
##             [,1]        [,2]
## [1,] 7.589615214 7.538461218
## [2,] 7.589615214 7.538461218
## [3,] 7.589615214 7.538461218
## [4,] 7.589615214 7.538461218
##
## $generalized.weights[[8]]
##             [,1]        [,2]
## [1,] 8.065688089 8.112470698
## [2,] 8.065688089 8.112470698
## [3,] 8.065688089 8.112470698
## [4,] 8.065688089 8.112470698
##
## $generalized.weights[[9]]
##             [,1]        [,2]
## [1,] 8.689400031 8.705870289
## [2,] 8.689400031 8.705870289
## [3,] 8.689400031 8.705870289
## [4,] 8.689400031 8.705870289
##
## $generalized.weights[[10]]
##             [,1]        [,2]
## [1,] 8.443097287 8.361864596
## [2,] 8.443097287 8.361864596
## [3,] 8.443097287 8.361864596
## [4,] 8.443097287 8.361864596
##
##
```

```
## $result.matrix
##                                1              2              3
## error                0.04807521487  0.050779778185  0.047609791102
## reached.threshold    0.00994100910  0.008579133074  0.008565247527
## steps              120.00000000000 122.000000000000 135.000000000000
## Intercept.to.AND   -12.40282877212 -12.286306228183 -12.444641076586
## Var1.to.AND          8.15652724988   8.067902390258   8.216079893034
## Var2.to.AND          8.19731798958   8.003712917911   8.155170355495
##                                 4                5                6
## error                0.056506453266   0.062604158862   0.047267117896
## reached.threshold    0.008970283249   0.009079661697   0.009277329182
## steps              126.000000000000 122.000000000000 133.000000000000
## Intercept.to.AND   -11.953921780015 -11.637935611335 -12.458067751025
## Var1.to.AND          7.789894471797   7.622987534182   8.203940689319
## Var2.to.AND          7.857669540719   7.613388632029   8.208728550577
##                                 7                8                9
## error                0.064306017509   0.050090330697   0.037053876321
## reached.threshold    0.009726227261   0.009768170626   0.007350999241
## steps              131.000000000000 129.000000000000 126.000000000000
## Intercept.to.AND   -11.557722160790 -12.283719423490 -13.192598827666
## Var1.to.AND          7.589615213684   8.065688088776   8.689400030858
## Var2.to.AND          7.538461218275   8.112470697835   8.705870289229
##                               10
## error                0.04263689822
## reached.threshold    0.00698589587
## steps              130.00000000000
## Intercept.to.AND   -12.78689065899
## Var1.to.AND          8.44309728741
## Var2.to.AND          8.36186459643
##
## attr(,"class")
## [1] "nn"
```

Now to validate the predictions:

```
input <- data.frame(expand.grid(c(0,1), c(0,1)))
net.results <- compute(net, input)
cbind(round(net.results$net.result), AND)
```

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