Neural Network in R

Saul Garcia

October 21, 2015

Neural Network in R

Summary

The objective of this project is to learn how to do a Neural Network. The objective is to generate a random dataset with the datagenerator file provided in the course, and with this dataset, build a supervised learning Neural Network that takes 4 variables as an input and predicts the output we are given. In this scenario, with the help of the Neuralnet package, it was possible to build this Neural Network which is able to predict with an accuracy of 100% and a confidence interval of 95%.

Loading the packages

```
library("neuralnet")

## Loading required package: grid
## Loading required package: MASS

library("caret")

## Loading required package: lattice
## Loading required package: ggplot2

set.seed(1234)

#Set working directory
setwd("/Users/saulgarcia/Dropbox/Maestría/DMKM/Courses/SEM1/Optimization/Laboratory/NeuralNetworkR")
```

Import and Clean the Data

The data generated is imported, and cleaned by removing some columns that are not useful for the problem, in order to end up with just our Input (X1,X2,X3,X4) and Output (Y). To do this, we are utilizing the split datasets "data.x" and "data_y" that were obtained by the code given in class.

The data is a [99x5] data frame which means: 99 observations for 4 Inputs and 1 Output. In order to fulfill our objective in this project, the 80% of our data will be used as training and 20% as testing.

```r
#Input
X <- read.csv("data_x.txt",sep = " ")[ ,2:5]
names(X)<- c("X1","X2","X3","X4")
#Output
Y = read.csv("data_y.txt",sep = " ")[,2]

#Bind in one dataframe
data = cbind(X,Y)
head(data)
```

```
##      X1    X2    X3    X4 Y
## 1 0.178 0.714 0.839 0.165 0
## 2 0.887 0.714 0.457 0.452 1
## 3 0.624 0.663 0.291 0.430 1
## 4 0.670 0.312 0.796 0.121 0
## 5 0.312 0.254 0.339 0.466 0
## 6 0.080 0.320 0.653 0.110 0
```

Neural Network Procedure

The previous dataset will be run through the neural network function obtained from the NeuralNet library. The neural network is going utilize an Error Back Propagation method, a supervised learning method, where it is taking the first four columns of the dataset as an input, and by using a sigmoidal function, it will iterate by utilizing a gradient descent. This means our model has to be differentiable. The model will utilize our known Output (Y column), in order to optimize and build the most accurate model of prediction, acording to our data.

As previously said, our neural network will be run with the last 79 observations, in order to use the first 20 observations to test. Since this data was generated randomly, it is basically the same if for this scenario the data partition is not randomized.

```r
#Train the neural network
#Going to have 2 hidden layers
#Threshold is a numeric value specifying the threshold for the partial
derivatives of the error function as stopping criteria.
  net.sqrt <- neuralnet(Y~X1+X2+X3+X4,data[21:99,], hidden=2, threshold=0.05)
  net.sqrt$result.matrix
```

```
##                                     1
## error                      0.33170463162
## reached.threshold          0.04819916932
## steps                   1642.00000000000
## Intercept.to.1layhid1     -5.49529826621
## X1.to.1layhid1             2.71120241352
## X2.to.1layhid1             3.27593380384
## X3.to.1layhid1             2.14659092376
## X4.to.1layhid1             3.20419439144
## Intercept.to.1layhid2     17.49453081533
## X1.to.1layhid2            -8.37075789831
```

```
## X2.to.1layhid2          -9.99805154492
## X3.to.1layhid2          -7.37516527169
## X4.to.1layhid2          -9.48379530261
## Intercept.to.Y           1.66735021116
## 1layhid.1.to.Y          -0.72042625951
## 1layhid.2.to.Y          -1.59494901264
```

This is giving us the number of iterations taken in order to find our optimal value. Also shows us our 10 optimized weights that were found for each hidden layer. Four weights for the first hidden layer and its intercept as an input, another four for the second hidden layer, and ultimately the last two weights and an intercept to input in order to obtain our prediction.

```
#Plot the neural network
  plot(net.sqrt)
#the plot will be attatched in a different document.
```

Testing

After having our data trained, its time to proove weather our Neural Netowk has learned, and how good it did. As it was stated, for our testing, only the first 20 observations will be used for our prediction.

```
#The code is computing our model built with the training set, with our
testing set (first 20 observations)
results <- compute(net.sqrt, X[1:20,])
```

Results

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

```
##              [,1]
## 1   0.16789183239
## 2   1.04629646136
## 3   0.66924890467
## 4   0.04696083469
## 5  -0.04196935462
## 6   0.01719785471
## 7  -0.06270959151
## 8   1.03542078737
## 9   1.00307134139
## 10  1.05247185274
## 11 -0.08213942190
## 12  1.02331500727
## 13 -0.05932484821
## 14 -0.00115191016
## 15 -0.05971468857
## 16 -0.01360722064
## 17  1.05341293740
## 18  1.06653505064
```

```
## 19  1.03919177102
## 20  0.05109261708

prediction<-as.data.frame(round(results$net.result))
```

The result we observe are still needed to go through the threshold function, where they will be rounded in order to have only 0 or 1 as an output.

After getting our predicted output values through the threshold function, we will be building a dataframe containing our Inputs, and Original or Expected Outputs, compared with our Predicted or Neural Network Outputs. Now our Output dataset (Y) is being transformed as a data frame, and being subset for utilizing the first 20 observations.

```
#First we subset our desired elements from our original outputs.
Ydf<-as.data.frame(Y[1:20])

resultsdf<-cbind(X[1:20,], Ydf , prediction)
names(resultsdf)<-c("I_X1","I_X2", "I_X3", "I_X4", "Expected Output" ,
"Neural Network Output")
#Print the results dataframe
resultsdf
```

```
##     I_X1  I_X2  I_X3  I_X4 Expected Output Neural Network Output
## 1  0.178 0.714 0.839 0.165               0                     0
## 2  0.887 0.714 0.457 0.452               1                     1
## 3  0.624 0.663 0.291 0.430               1                     1
## 4  0.670 0.312 0.796 0.121               0                     0
## 5  0.312 0.254 0.339 0.466               0                     0
## 6  0.080 0.320 0.653 0.110               0                     0
## 7  0.041 0.844 0.649 0.064               0                     0
## 8  0.396 0.970 0.137 0.690               1                     1
## 9  0.202 0.637 0.476 0.866               1                     1
## 10 0.460 0.424 0.684 0.779               1                     1
## 11 0.380 0.885 0.046 0.230               0                     0
## 12 0.089 0.549 0.746 0.862               1                     1
## 13 0.570 0.228 0.557 0.221               0                     0
## 14 0.109 0.807 0.113 0.049               0                     0
## 15 0.032 0.145 0.810 0.722               0                     0
## 16 0.017 0.276 0.848 0.242               0                     0
## 17 0.758 0.538 0.462 0.673               1                     1
## 18 0.263 0.798 0.858 0.446               1                     1
## 19 0.570 0.438 0.904 0.433               1                     1
## 20 0.161 0.241 0.084 0.187               0                     0
```

With this printing we can proove that our predictions are exactly the same as our expected output values from our generated dataset, but in order to represent it graphically we have to do a Confusion Matrix.

```
confusionMatrix(prediction$V1, Ydf$`Y[1:20]`)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 11  0
##          1  0  9
##
##                Accuracy : 1
##                  95% CI : (0.8315665, 1)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 0.000006415844
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.00
##             Specificity : 1.00
##          Pos Pred Value : 1.00
##          Neg Pred Value : 1.00
##              Prevalence : 0.55
##          Detection Rate : 0.55
##    Detection Prevalence : 0.55
##       Balanced Accuracy : 1.00
##
##        'Positive' Class : 0
##
```

As conclusion, we have developed neural network model that utilizing four real variables, predicts with a 100% of accuracy a binary output. It is safe to say that our model works, given that we obtained a 95% confidence level four our study.