Neural Networks

João Neto

May 2013

A neural network (NN) model is very similar to a non-linear regression model, with the exception that the former can handle an incredibly large amount of model parameters. For this reason, neural network models are said to have the ability to approximate any continuous function.

Neural network is very good at learning non-linear function and also multiple outputs can be learnt at the same time. However, the training time is relatively long and it is also susceptible to local minimum traps. This can be mitigated by doing multiple rounds and pick the best learned model.

R has several packages for dealing with NNs, like `neuralnet` , `nnet` or `RSNNS` .

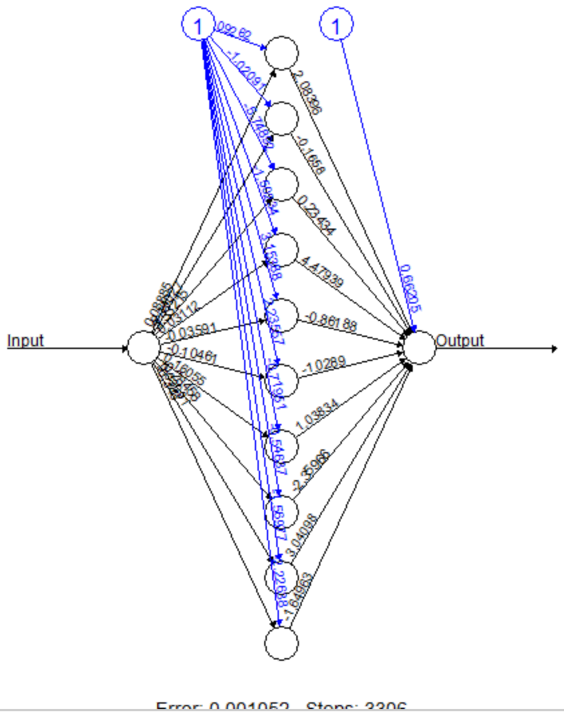
Using package neuralnet

From (http://gekkoquant.com/2012/05/26/neural-networks-with-r-simple-example/)http://gekkoquant.com/2012/05/26/neural-networks-with-r-simple-example/ (http://gekkoquant.com/2012/05/26/neural-networks-with-r-simple-example/)

```
library(neuralnet)

#Going to create a neural network to perform square rooting (using backpropagation)
#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 with backpropagation
#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,     thre
shold = 0.01)
##
## 1 repetition was calculated.
##
##              Error Reached Threshold Steps
## 1 0.0006282191137    0.009245995019   6812
```

```
#Plot the neural network
plot(net.sqrt)
```



```
#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,] 0.9143358513
##  [2,] 1.9942238404
##  [3,] 3.0005677587
##  [4,] 3.9981018905
##  [5,] 5.0008748616
##  [6,] 6.0022207302
##  [7,] 6.9938605784
##  [8,] 7.9987651833
##  [9,] 9.0110881921
## [10,] 9.9793247077
```

```
#Lets display a better version of the results
cleanoutput <- cbind(testdata,sqrt(testdata),
                         as.data.frame(net.results$net.result))
colnames(cleanoutput) <- c("Input","Expected Output","Neural Net Output")
print(cleanoutput)
```

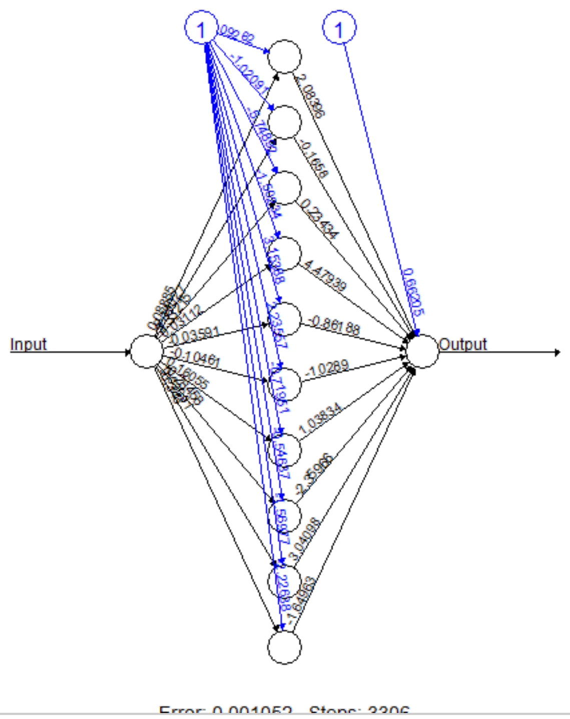
```
##    Input Expected Output Neural Net Output
## 1      1               1      0.9143358513
## 2      4               2      1.9942238404
## 3      9               3      3.0005677587
## 4     16               4      3.9981018905
## 5     25               5      5.0008748616
## 6     36               6      6.0022207302
## 7     49               7      6.9938605784
## 8     64               8      7.9987651833
## 9     81               9      9.0110881921
## 10   100              10      9.9793247077
```

Another example (http://horicky.blogspot.pt/2012/06/predictive-analytics-neuralnet-bayesian.html)

```
library(neuralnet)

set.seed(101)
size.sample <- 50
iristrain <- iris[sample(1:nrow(iris), size.sample),] # get a training sample from iris
nnet_iristrain <- iristrain

# Binarize the categorical output
nnet_iristrain <- cbind(nnet_iristrain, iristrain$Species == 'setosa')
nnet_iristrain <- cbind(nnet_iristrain, iristrain$Species == 'versicolor')
nnet_iristrain <- cbind(nnet_iristrain, iristrain$Species == 'virginica')

names(nnet_iristrain)[6] <- 'setosa'
names(nnet_iristrain)[7] <- 'versicolor'
names(nnet_iristrain)[8] <- 'virginica'

head(nnet_iristrain)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species setosa
## 56           5.7         2.8          4.5         1.3 versicolor  FALSE
## 7            4.6         3.4          1.4         0.3     setosa   TRUE
## 106          7.6         3.0          6.6         2.1  virginica  FALSE
## 97           5.7         2.9          4.2         1.3 versicolor  FALSE
## 37           5.5         3.5          1.3         0.2     setosa   TRUE
## 44           5.0         3.5          1.6         0.6     setosa   TRUE
##     versicolor virginica
## 56        TRUE     FALSE
## 7        FALSE     FALSE
## 106      FALSE      TRUE
## 97        TRUE     FALSE
## 37       FALSE     FALSE
## 44       FALSE     FALSE
```

```
nn <- neuralnet(setosa+versicolor+virginica ~
                Sepal.Length+Sepal.Width
                            +Petal.Length
                            +Petal.Width,
                data=nnet_iristrain,
                hidden=c(3))

plot(nn)
```



```
mypredict <- compute(nn, iris[-5])$net.result
# Put multiple binary output to categorical output
maxidx <- function(arr) {
    return(which(arr == max(arr)))
}
idx <- apply(mypredict, c(1), maxidx)
prediction <- c('setosa', 'versicolor', 'virginica')[idx]
table(prediction, iris$Species)
```

```
##
## prediction   setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         47         3
##   virginica       0          3        47
```

Using package nnet

From (http://www.r-bloggers.com/visualizing-neural-networks-from-the-nnet-package/)http://www.r-bloggers.com/visualizing-neural-networks-from-the-nnet-package/ (http://www.r-bloggers.com/visualizing-neural-networks-from-the-nnet-package/)

Let's use the iris dataset as an example:

```
library(nnet)

ir      <- rbind(iris3[,,1], iris3[,,2], iris3[,,3])
targets <- class.ind( c(rep("s", 50), rep("c", 50), rep("v", 50)) )
samp    <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
nn.iris <- nnet(ir[samp,], targets[samp,], size = 2, rang = 0.1,decay = 5e-4, maxit = 2
00)
```

```
## # weights:  19
## initial  value 55.469165
## iter  10 value 25.300941
## iter  20 value 25.173359
## iter  30 value 25.111613
## iter  40 value 22.372522
## iter  50 value 17.998055
## iter  60 value 17.250091
## iter  70 value 16.985991
## iter  80 value 10.161571
## iter  90 value 1.280245
## iter 100 value 0.922433
## iter 110 value 0.702587
## iter 120 value 0.597699
## iter 130 value 0.555386
## iter 140 value 0.532015
## iter 150 value 0.525445
## iter 160 value 0.524072
## iter 170 value 0.522200
## iter 180 value 0.521476
## iter 190 value 0.520531
## iter 200 value 0.520347
## final  value 0.520347
## stopped after 200 iterations
```

This is a plot function for nnet neural nets called `plotnet`. Read (https://beckmw.wordpress.com/2013/11/14/visualizing-neural-networks-in-r-update/)https://beckmw.wordpress.com/2013/11/14/visualizing-neural-networks-in-r-update/

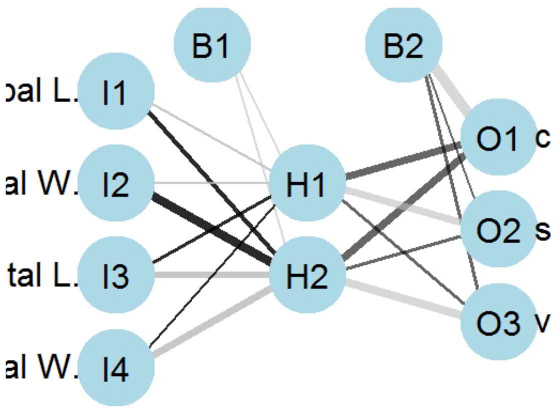
(https://beckmw.wordpress.com/2013/11/14/visualizing-neural-networks-in-r-update/) for visualizing options.

Here are some:

- mod.in model object for input created from nnet function
- nid logical value indicating if neural interpretation diagram is plotted, default T
- all.out logical value indicating if all connections from each response variable are plotted, default T
- all.in logical value indicating if all connections to each input variable are plotted, default T
- wts.only logical value indicating if connections weights are returned rather than a plot, default F
- rel.rsc numeric value indicating maximum width of connection lines, default 5
- circle.cex numeric value indicating size of nodes, passed to cex argument, default 5
- node.labs logical value indicating if text labels are plotted, default T
- line.stag numeric value that specifies distance of connection weights from nodes
- cex.val numeric value indicating size of text labels, default 1
- alpha.val numeric value (0-1) indicating transparency of connections, default 1
- circle.col text value indicating color of nodes, default 'lightgrey'
- pos.col text value indicating color of positive connection weights, default 'black'
- neg.col text value indicating color of negative connection weights, default 'grey'

Now let's plot the previous trained neural net:

```
library(NeuralNetTools)

plotnet(nn.iris, alpha=0.6)
```



Using NNs via the Caret package

The caret package (short for classification and regression training) contains functions to streamline the model training process for complex regression and classification problems.

Check (http://caret.r-forge.r-project.org/modelList.html)http://caret.r-forge.r-project.org/modelList.html (http://caret.r-forge.r-project.org/modelList.html) for a list of available methods.

```
library(quantmod)
```

```
## Loading required package: xts
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## Loading required package: TTR
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
library(nnet)
library(caret)
```

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

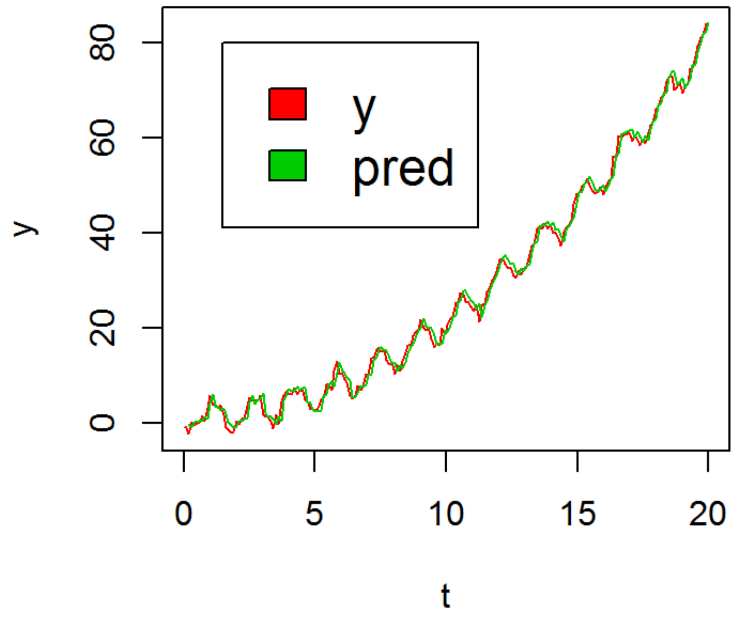
```
t <- seq(0,20,length=200)                       # time stamps
y <- 1 + 3*cos(4*t+2) +.2*t^2 + rnorm(200)      # the time series we want to predict
dat <- data.frame( y, x1=Lag(y,1), x2=Lag(y,2)) # create a triple with lagged values
names(dat) <- c('y','x1','x2')
head(dat)
```

```
##               y            x1            x2
## 1 -1.0628369855            NA            NA
## 2 -0.9461638315 -1.0628369855            NA
## 3 -2.4148119350 -0.9461638315 -1.0628369855
## 4  0.1578938481 -2.4148119350 -0.9461638315
## 5 -0.4744434653  0.1578938481 -2.4148119350
## 6 -0.1407492830 -0.4744434653  0.1578938481
```

```
#Fit model
model <- train(y ~ x1+x2, dat, method='nnet', linout=TRUE, trace = FALSE)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```
ps <- predict(model, dat)

#Examine results

plot(t,y,type="l",col = 2)
lines(t[-c(1:2)],ps, col=3)
legend(1.5, 80, c("y", "pred"), cex=1.5, fill=2:3)
```



Caret is also able to make train/test sets. Using Caret for the isis dataset:

```
inTrain <- createDataPartition(y=iris$Species, p=0.75, list=FALSE)   # We wish 75% for
the trainset

train.set <- iris[inTrain,]
test.set  <- iris[-inTrain,]
nrow(train.set)/nrow(test.set) # should be around 3
```

```
## [1] 3.166666667
```

```
model <- train(Species ~ ., train.set, method='nnet', trace = FALSE) # train
# we also add parameter 'preProc = c("center", "scale"))' at train() for centering and
scaling the data
prediction <- predict(model, test.set[-5])                           # predict
table(prediction, test.set$Species)                                  # compare
```

```
## 
## prediction   setosa versicolor virginica
##   setosa         12          0         0
##   versicolor      0         12         1
##   virginica       0          0        11
```

```
# predict can also return the probability for each class:
prediction <- predict(model, test.set[-5], type="prob")
head(prediction)
```

```
##          setosa     versicolor      virginica
## 3  0.9833909273 0.012318908806 0.004290163909
## 13 0.9804309544 0.014774233586 0.004794812024
## 17 0.9876425117 0.008858596249 0.003498892061
## 30 0.9766536305 0.017949582649 0.005396786813
## 31 0.9751143511 0.019253775537 0.005631873333
## 34 0.9884559227 0.008206873188 0.003337204088
```