Lesson: Introduction to Neural Network in R

Introduction

Neural network is a powerful model inspired by how the brain works. It can perform regression tasks as well as classification tasks. A multilayer neural network contains one input layer, one or more hidden layers and one output layer. Compared to Linear Regression model, neural networks can learn nonlinearity relationship of independent variables from training data. Neural network models have nice theoretical properties and been applied to many real-world problems such as autonomous driving.

Fig.1 below shows the structure of a simple neural network with one hidden layer. This model has an input layer with 3 input nodes, a hidden layer with 4 hidden nodes and an output layer with a result node. These nodes are interconnected by weights (links between nodes of different layers). These weights will form a weight matrix learned from training data. There is a bias node (dotted circle) linked to each node other than input nodes and hidden nodes.

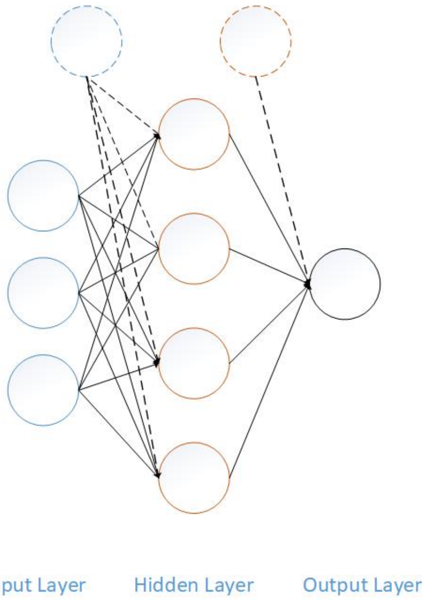


Figure 1. basic structure of neural network

Fig.2 illustrates how a single neural node computes the output. Let’s take a hidden node as an example. The input value of hidden node (*node*$_j$) is the weighted sum of all input nodes plus the bias node (1), which can be represented as

$$node_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0}$$

, where the subscript *i* denotes nodes of the input layer (previous layer), *j* denotes the hidden node, w_{ji} denotes the input-to-hidden weights of hidden node *j*. Each hidden node produces an output value a_j from its input value using a non-linear activation function, which might be a sigmoid function or ReLU function.

$$a_j = \delta(node_j)$$

The output of the node in output layer (the predicted value) is also the weighted sum of all hidden nodes in the previous layer plus the bias node (1).

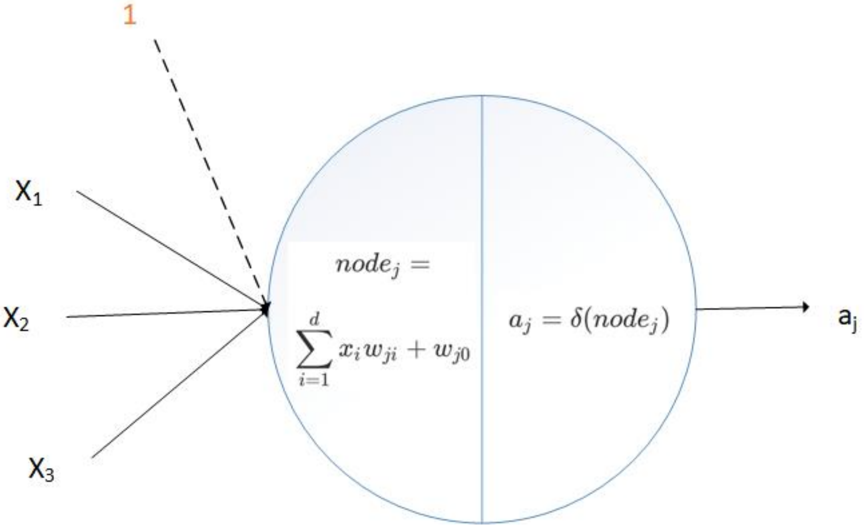


Figure 2. non-linear transformation of a hidden neural node

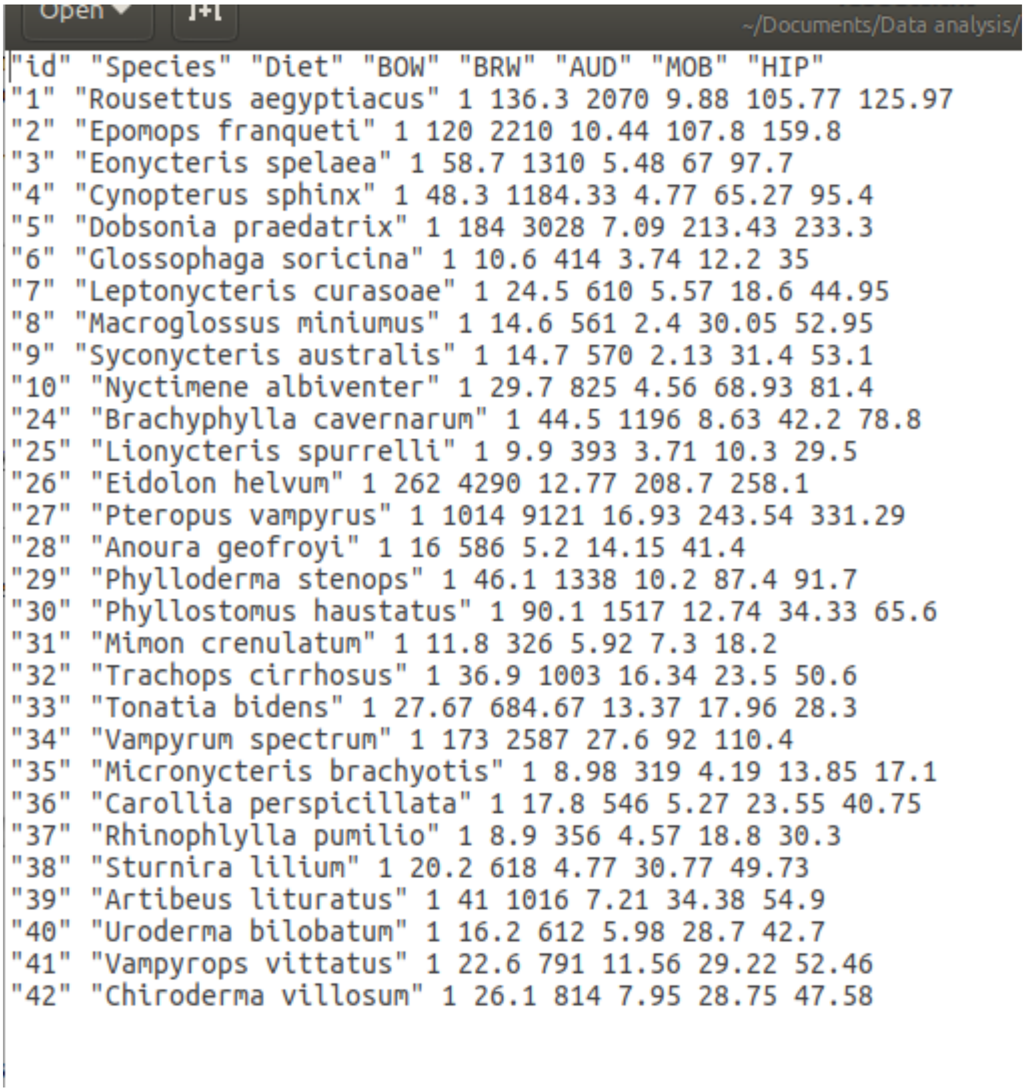
Gradient descent algorithms like Backpropagation are popular methods for training the weight matrix for multilayer neural networks.

In this lesson, we will learn how to use a neural network in R for regression purpose and to predict some information from a dataset.

Using a neural network model in a study can be decomposed into several steps :

- Find a tool to explore the dataset and to use the neural network model
- Clean the dataset and keep only the useful data
- Normalize each column in the dataset which contains the values of a certain independent or dependent variable
- Sample the dataset, divide it into training set and test set
- Train the weight matrix of neural network model using training data
- Correct the model if necessary
- Run the neural network model to predict the value of dependent variable in test data
- Denormalize the predicted value and compare it to the ground-truth value
- Make some conclusion on the model and its performance

In this example, we are going to use the following dataset:

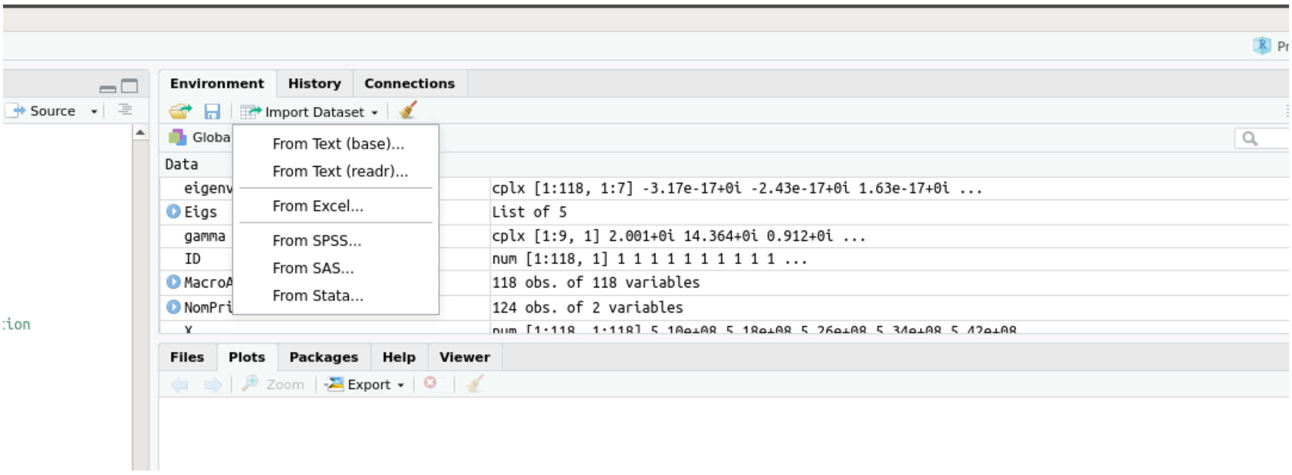


```
"id" "Species" "Diet" "BOW" "BRW" "AUD" "MOB" "HIP"
"1" "Rousettus aegyptiacus" 1 136.3 2070 9.88 105.77 125.97
"2" "Epomops franqueti" 1 120 2210 10.44 107.8 159.8
"3" "Eonycteris spelaea" 1 58.7 1310 5.48 67 97.7
"4" "Cynopterus sphinx" 1 48.3 1184.33 4.77 65.27 95.4
"5" "Dobsonia praedatrix" 1 184 3028 7.09 213.43 233.3
"6" "Glossophaga soricina" 1 10.6 414 3.74 12.2 35
"7" "Leptonycteris curasoae" 1 24.5 610 5.57 18.6 44.95
"8" "Macroglossus miniumus" 1 14.6 561 2.4 30.05 52.95
"9" "Syconycteris australis" 1 14.7 570 2.13 31.4 53.1
"10" "Nyctimene albiventer" 1 29.7 825 4.56 68.93 81.4
"24" "Brachyphylla cavernarum" 1 44.5 1196 8.63 42.2 78.8
"25" "Lionycteris spurrelli" 1 9.9 393 3.71 10.3 29.5
"26" "Eidolon helvum" 1 262 4290 12.77 208.7 258.1
"27" "Pteropus vampyrus" 1 1014 9121 16.93 243.54 331.29
"28" "Anoura geofroyi" 1 16 586 5.2 14.15 41.4
"29" "Phylloderma stenops" 1 46.1 1338 10.2 87.4 91.7
"30" "Phyllostomus haustatus" 1 90.1 1517 12.74 34.33 65.6
"31" "Mimon crenulatum" 1 11.8 326 5.92 7.3 18.2
"32" "Trachops cirrhosus" 1 36.9 1003 16.34 23.5 50.6
"33" "Tonatia bidens" 1 27.67 684.67 13.37 17.96 28.3
"34" "Vampyrum spectrum" 1 173 2587 27.6 92 110.4
"35" "Micronycteris brachyotis" 1 8.98 319 4.19 13.85 17.1
"36" "Carollia perspicillata" 1 17.8 546 5.27 23.55 40.75
"37" "Rhinophlylla pumilio" 1 8.9 356 4.57 18.8 30.3
"38" "Sturnira lilium" 1 20.2 618 4.77 30.77 49.73
"39" "Artibeus lituratus" 1 41 1016 7.21 34.38 54.9
"40" "Uroderma bilobatum" 1 16.2 612 5.98 28.7 42.7
"41" "Vampyrops vittatus" 1 22.6 791 11.56 29.22 52.46
"42" "Chiroderma villosum" 1 26.1 814 7.95 28.75 47.58
```

This dataset is composed of 7 variables: Diet, body masses (BOW), brain masses (BRW) and volumes of three brain regions (main olfactory bulb MOB, hippocampus HIP, auditory nuclei AUD) for 29 bats. In this example, we are trying to predict the value of BRW using the value of BOW, MOB, HIP and AUD as inputs.

Step 1: Load the dataset

First you have to install RStudio and then import the dataset.



```
Data
  eigenv     cplx [1:118, 1:7] -3.17e-17+0i -2.43e-17+0i 1.63e-17+0i ...
  Eigs       List of 5
  gamma      cplx [1:9, 1] 2.001+0i 14.364+0i 0.912+0i ...
  ID         num [1:118, 1] 1 1 1 1 1 1 1 1 1 1 ...
  MacroA     118 obs. of 118 variables
  NomPri     124 obs. of 2 variables
  X          num [1:118, 1:118] 5.10e+08 5.18e+08 5.26e+08 5.34e+08 5.42e+08
```

This will copy the dataset in a data frame variable called “Tabbats”. The same effect can be obtained by running the command (shown in terminal window):

```
Tabbats <- read.table("Tabbats.txt", header = TRUE,
                      stringsAsFactors = FALSE)
```

Step 2: Clean the dataset

In this step, you have to remove the data not useful for a regression neural network model with the following command:

```
str(Tabbats)
Tabbats <- Tabbats[,(4:8)]
str(TabBats)
```

We removed 3 attributes:

- The ID because it’s irrelevant;
- Species because they are character values
- Diet, because they are all phytophagy(1).

To do that we use the following command: Tabbats <- Tabbats[,(4:8)] which removes the first 3 columns from data frame Tabbats.

The result is a new table Tabbats without the columns 1 to 3 of the original table. We are left with 5 variables: BOW, BRW, AUD. MOB and HIP.

Step 3: Normalize the data

When training a neural network, one of the techniques that will speed up the training process is to normalize the inputs. If the input variables (features) come from very different scales, maybe some variables are from 0 to 1, some from 1 to 1,000, the gradient descent algorithm might need a lot of steps to oscillate back and forth before it converges when we run it on the cost function of neural network model.

So we need to normalize the training and test data. This time we use the min-max normalization solution. We extract the maximum and minimum value from each column (variable) of table Tabbats, and use them to scale the variable values to the range [0, 1]. The following commands need to be executed:

```
maxs <- apply(Tabbats, 2, max)
mins <- apply(Tabbats, 2, min)
scaled <- as.data.frame(scale(Tabbats, center = mins, scale = maxs - mins))
```

The data set now becomes as follows:

```
          BOW          BRW          AUD          MOB          HIP
1  1.267536e-01 0.1989320609 0.30427954 0.41682188 0.346510074
2  1.105363e-01 0.2148375369 0.32626620 0.42541483 0.454183774
3  4.954731e-02 0.1125880482 0.13152729 0.25270911 0.256532671
4  3.920008e-02 0.0983106112 0.10365135 0.24538605 0.249212260
5  1.742115e-01 0.3077709611 0.19473891 0.87254487 0.688118654
6  1.691374e-03 0.0107930016 0.06321162 0.02074162 0.056971896
……
```

Step 4: Divide the data set into training and test data

You can shuffle the whole data set using sample() function. We extract 80% of the data as training data, and leave the remaining as test data. The following commands need to be executed:

```
indarr <- sample(1:nrow(scaled),round(0.8*nrow(scaled)))
train <- scaled[indarr,]
test <- scaled[-indarr,]
```

Step 5: Train the neural network model

At first, we create a neural network model with just one hidden layer, and this layer has ten hidden nodes. We use four independent variables BOW, AUD. MOB and HIP to constitute the input layer of neural network, and the node in output layer produces the value of dependent variable BRW.

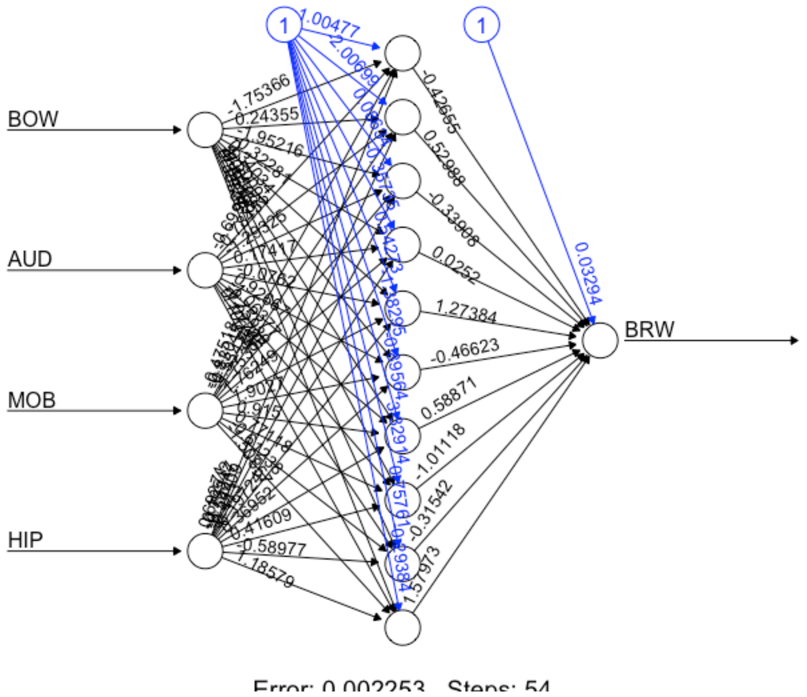
We use training data to train the model and you need to notice that the parameter linear.output should be set to ‘T’ so that the model performs a regression task other than a classification task.

The following commands need to be executed:

```
install.packages("neuralnet")
library(neuralnet)
f <- as.formula(BRW~BOW+AUD+MOB+HIP)
nn <- neuralnet(f,data=train,hidden=c(10),linear.output=T)
#train NN with normalized training data
```

Then we can use plot(nn) to illustrate the neural network structure we create:

```
plot(nn)
```



To check the trained weight matrix of the neural network model, we can use the following command:

```
print(nn$result.matrix)
```

It will show the training error, total number of training steps and weights learned through training process:

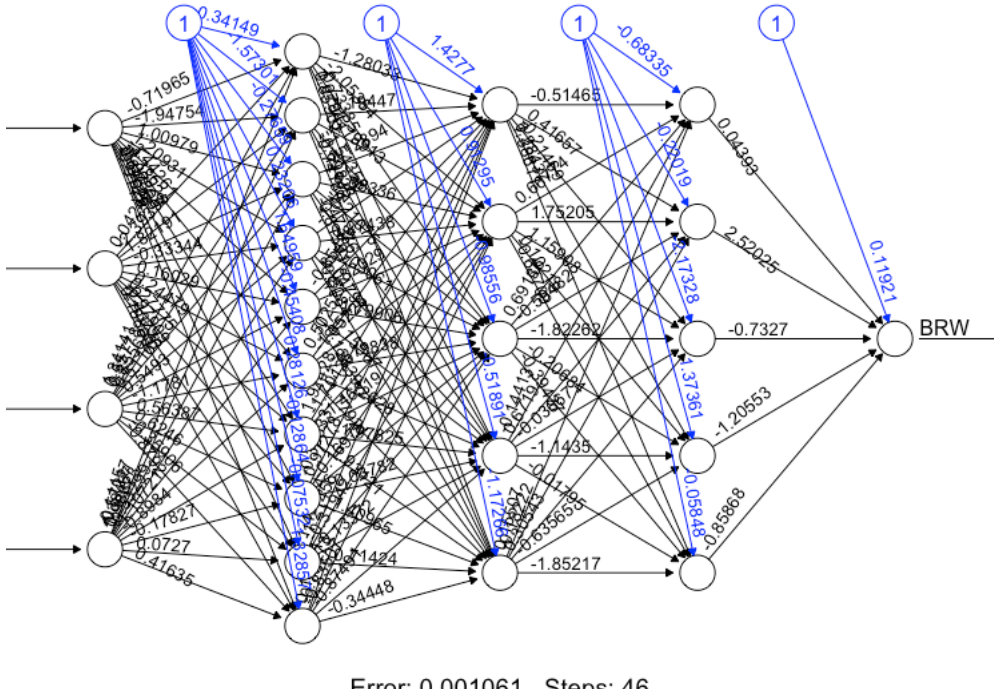
```
                              [,1]
error                      0.004328590
reached.threshold          0.007786513
steps                      46.000000000
Intercept.to.1layhid1  -0.966326161
BOW.to.1layhid1          0.109112501
AUD.to.1layhid1          1.465561442
MOB.to.1layhid1         -1.758688346
HIP.to.1layhid1        -1.364149755
Intercept.to.1layhid2   0.388015818
BOW.to.1layhid2          0.297679272
AUD.to.1layhid2          1.592182508
MOB.to.1layhid2         -0.167911954
HIP.to.1layhid2        -0.281130733
……
Intercept.to.1layhid10  0.960158512
BOW.to.1layhid10          0.681061421
AUD.to.1layhid10         -0.934404588
MOB.to.1layhid10          1.529719985
HIP.to.1layhid10         0.620069556
Intercept.to.BRW         0.776807052
1layhid1.to.BRW         -0.707508387
1layhid2.to.BRW         -0.949095913
1layhid3.to.BRW         -0.662794336
1layhid4.to.BRW         -1.911382964
1layhid5.to.BRW          0.875435148
1layhid6.to.BRW          0.483427145
1layhid7.to.BRW         -0.535594401
```

```
1layhid8.to.BRW      -0.999112625
1layhid9.to.BRW       1.072337799
1layhid10.to.BRW       0.480026323
```

Step 6: Correct the model if necessary

We can adjust the number of hidden layers and hidden nodes in the neural network model to further improve its prediction performance. This time we add two hidden layers in the above model using the following command, and the new structure is illustrated in the diagram below.

```
nn <- neuralnet(f,data=train,hidden=c(10,5,5),linear.output=T)
```



The training error could be reduced to 0.001 or less.

Step 7: Predict the value of dependent variable in test data

With the trained neural network, we can input the values of independent variables of records in test data and run the NN model to produce the predicted value of dependent variable. The following command needs to be executed:

```
predicted <- compute(nn,test)
```

Note that the predicted BRW values are still in their normalized form.

Step 8: Denormalize the result and compare it to the ground-truth value

We can use the reverse procedure of data normalization to restore the predicted BRW values and their corresponding ground truth values. The following commands need to be executed:

```
predicted_t <- predicted$net.result*(max(Tabbats$BRW)-
min(Tabbats$BRW))+min(Tabbats$BRW)  #result denormalizaiton
test_gt <- test$BRW*(max(Tabbats$BRW)-min(Tabbats$BRW))+min(Tabbats$BRW)
#ground truth value
print(paste(test_gt, predicted_t))
```

Similar comparison results would be showed in the console:

```
[1] "3028 3442.89066339023" "610 617.37974958612"   "393 400.682269239281"
[4] "1003 941.446060097756" "2587 2371.70002430478" "814 715.735029481705"
```

Step 9: Discuss the model's performance

You can find that the model's test error is higher than training error, that is called the overfitting problem. We can adjust the NN model structure (decrease the number of hidden layers and hidden nodes) or increase the number of training samples to reduce the test error.

Exercise

Repeat the analysis, adjust the data normalization method, train-test data ratio and the structure of NN model.