Neural Network Using Data on Graduate School Admission Data

Introduction:

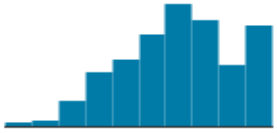
I created a basic neural networking in R using the package Neuralnet. My goal is to see if I can go machine learning to determine if a student got accepted to graduate school based on their GRE (graduate record examination), GPA (grade point average), and the ranking of the school they came from.

Neuralnet Package

Neuralnet uses backpropagation to train neural networks. Backpropagation is an algorithm for supervised learning of artificial neural networks using gradient descent. The “backwards” part is that the calculation proceeds backwards through the network with the final layer of weights being calculated first and the first layer being calculated last. The supervised learning algorithms are characterized by the usage of a given output that is compared to the predicted output and by the adaption of all parameters according to this comparison. The parameters are its weights which are initialized with random values drawn from a standard normal distribution.

The Dataset

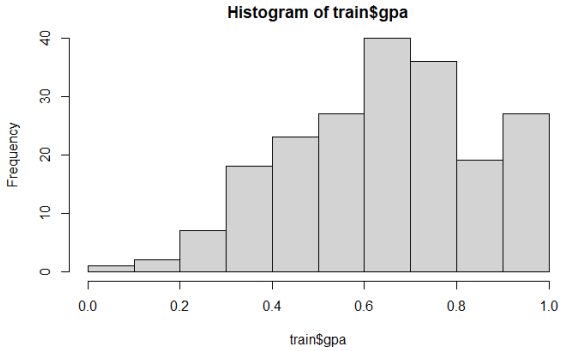
Data is used from *Kaggle*.

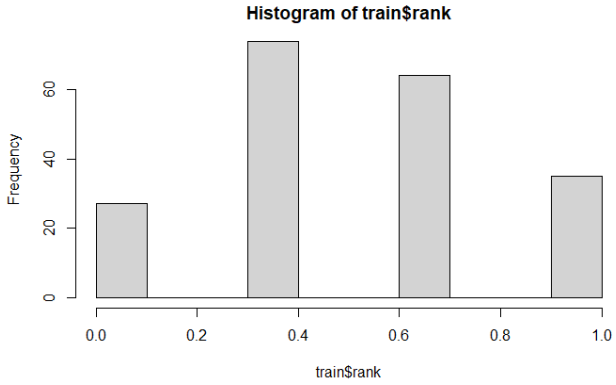
# admit	# gre	# gpa	# rank
			
0 1	220 800	2.26 4	1 4
0	380	3.61	3
1	660	3.67	3
1	800	4	1
1	640	3.19	4
0	520	2.93	4
1	760	3	2
1	560	2.98	1
0	400	3.08	2
1	540	3.39	3
0	700	3.92	2

I split the data into two different files, one training dataset and one test dataset. The training dataset has 200 students in the set and the test dataset has 99 students. I put more students in the training dataset so that the machine can learn more and possibly provide better results.

The Coding Process

I first imported the Neuralnet package and imported the training dataset. I noticed that there were some problems when I tried making a neural network off the bat, so I had to normalize all my data points so that they are all points between 0 and 1.

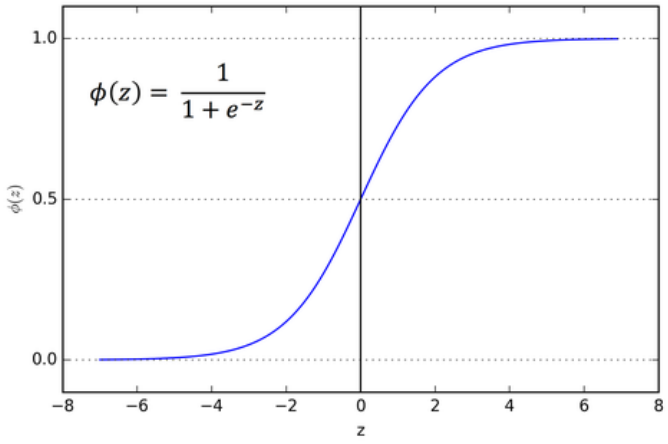




After normalizing the data, I am now able to create a neural network.

$$\mathrm{E}_{\mathrm{sse}} = \frac{1}{2} \sum_{l=1}^{L} \sum_{h=1}^{H} (o_{lh} - y_{lh})^2$$

The error function used in the neural network is the sum of squares error (SSE) which is the summed difference between the observed value and the predicted values squared. L is the indexes observed, H is the output nodes, o is the predicted output, and y is the observed output.

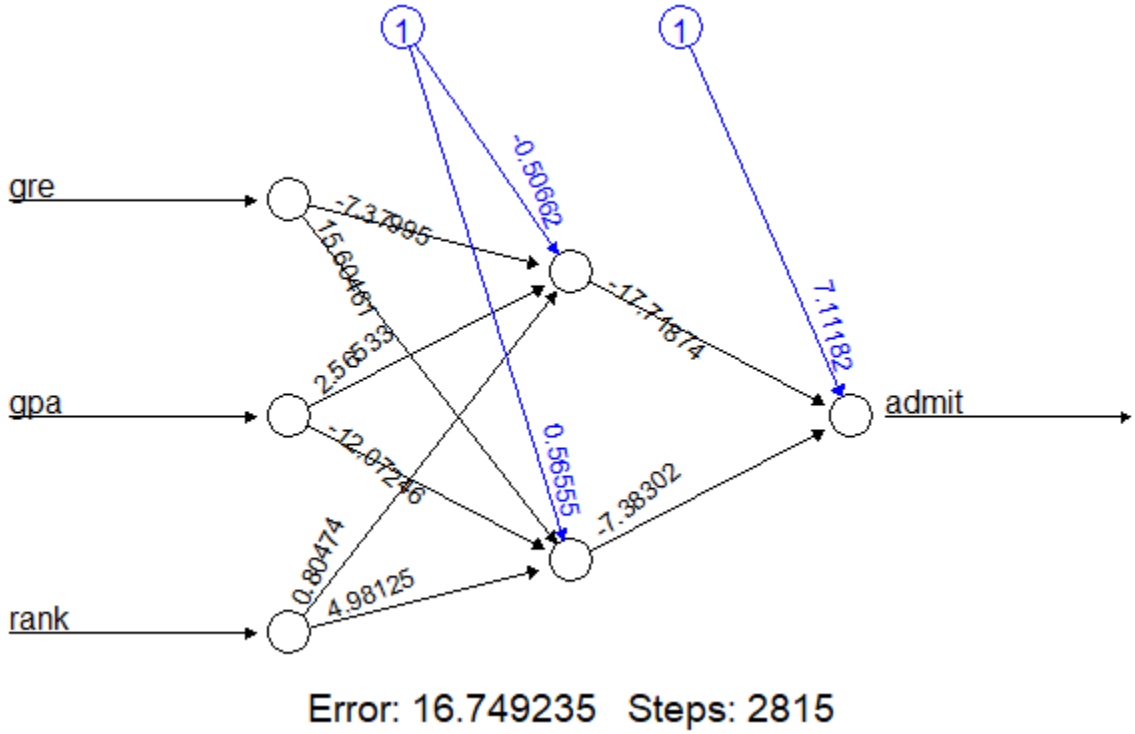


The activation function used in the neural network is a logistic function. The activation function transforms aggregated input signals into output signals.

The repetition function in Neuralnet runs the neural network x number of times and returns the best output (the one with the lowest error). I did five repetitions.

After getting the neural network, I imported the test dataset and tried to see if it could figure out if the student got admitted based on GRE, GPA, and school rank. Unlike the training dataset, the test dataset does not include the “admit” variable. I would then run a prediction based on my results of the neural network on my test data and get probability outputs on whether the student passed or not. To make it easier to read the results, I converted the probabilities into binary form to where if the probability was over 50%, it will be a 1 meaning pass and if not, then a 0 meaning fail.

Results

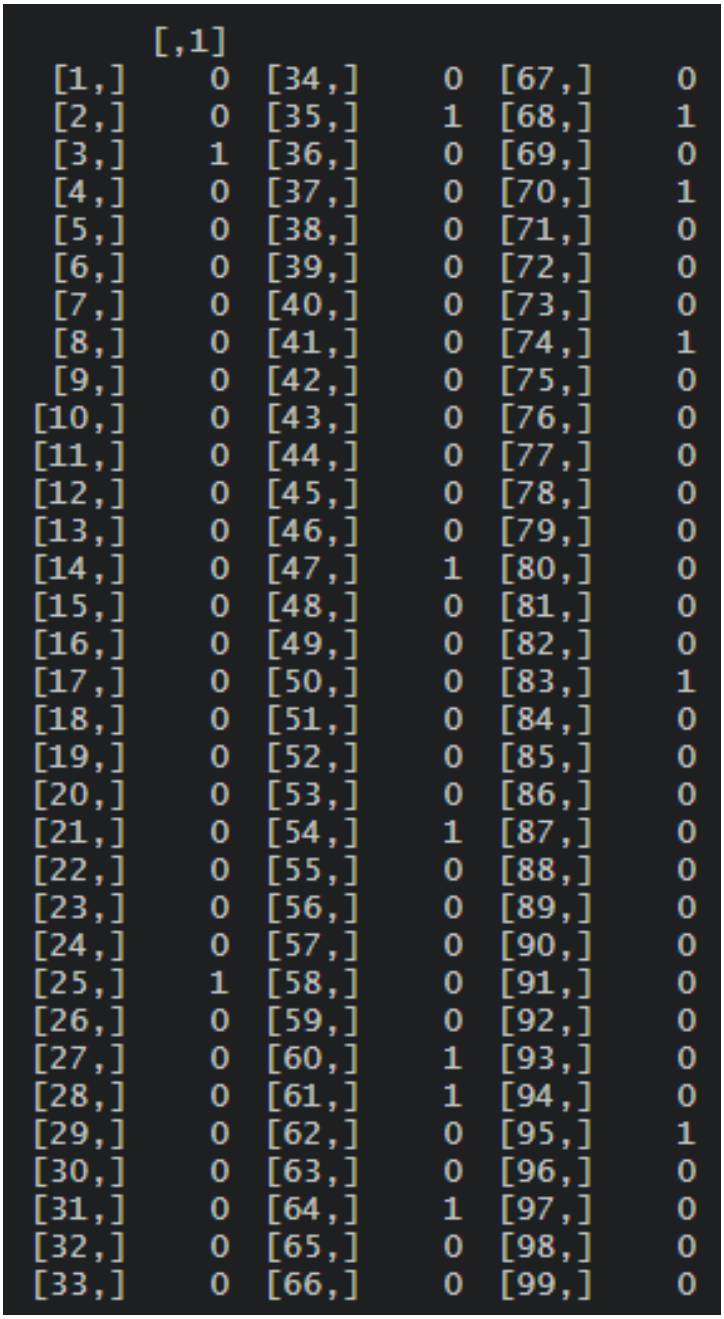


Error: 16.749235 Steps: 2815

Probabilities:

```
          [,1]                                                  
 [1,] 0.1621871  [34,] 0.1622319  [67,] 0.1621402
 [2,] 0.1622971  [35,] 0.8034128  [68,] 0.8217415
 [3,] 0.5725166  [36,] 0.1621266  [69,] 0.4298304
 [4,] 0.1621267  [37,] 0.1621266  [70,] 0.5138132
 [5,] 0.1621266  [38,] 0.4437439  [71,] 0.1629157
 [6,] 0.4360947  [39,] 0.1621464  [72,] 0.4197876
 [7,] 0.4183214  [40,] 0.1621266  [73,] 0.4804059
 [8,] 0.2084135  [41,] 0.1621266  [74,] 0.5039077
 [9,] 0.1621266  [42,] 0.1621352  [75,] 0.1621266
[10,] 0.1647755  [43,] 0.3189273  [76,] 0.4250486
[11,] 0.4246633  [44,] 0.1621274  [77,] 0.4606560
[12,] 0.1623949  [45,] 0.1621266  [78,] 0.1621269
[13,] 0.1621267  [46,] 0.4160711  [79,] 0.1635701
[14,] 0.1621266  [47,] 0.5015727  [80,] 0.4217907
[15,] 0.1621389  [48,] 0.1648111  [81,] 0.3796386
[16,] 0.1621822  [49,] 0.3590922  [82,] 0.3863124
[17,] 0.1621266  [50,] 0.4741179  [83,] 0.8212059
[18,] 0.1621270  [51,] 0.1621282  [84,] 0.1621443
[19,] 0.4835772  [52,] 0.1621266  [85,] 0.4343559
[20,] 0.1621267  [53,] 0.4201407  [86,] 0.4368506
[21,] 0.1981339  [54,] 0.5249995  [87,] 0.4080026
[22,] 0.1621266  [55,] 0.4197918  [88,] 0.4183071
[23,] 0.1621276  [56,] 0.4197848  [89,] 0.4205004
[24,] 0.1621266  [57,] 0.4261231  [90,] 0.4197874
[25,] 0.8180828  [58,] 0.1621694  [91,] 0.4896386
[26,] 0.4197789  [59,] 0.1996390  [92,] 0.1621271
[27,] 0.4066062  [60,] 0.8218715  [93,] 0.4453007
[28,] 0.4058304  [61,] 0.7356159  [94,] 0.1731473
[29,] 0.1621266  [62,] 0.4196305  [95,] 0.6975103
[30,] 0.2932887  [63,] 0.1702273  [96,] 0.1621266
[31,] 0.1621821  [64,] 0.5373571  [97,] 0.1621322
[32,] 0.4263703  [65,] 0.1621266  [98,] 0.4217907
[33,] 0.1621279  [66,] 0.1621266  [99,] 0.1624689
```

Binary Form:



[,1]					
[1,]	0	[34,]	0	[67,]	0
[2,]	0	[35,]	1	[68,]	1
[3,]	1	[36,]	0	[69,]	0
[4,]	0	[37,]	0	[70,]	1
[5,]	0	[38,]	0	[71,]	0
[6,]	0	[39,]	0	[72,]	0
[7,]	0	[40,]	0	[73,]	0
[8,]	0	[41,]	0	[74,]	1
[9,]	0	[42,]	0	[75,]	0
[10,]	0	[43,]	0	[76,]	0
[11,]	0	[44,]	0	[77,]	0
[12,]	0	[45,]	0	[78,]	0
[13,]	0	[46,]	0	[79,]	0
[14,]	0	[47,]	1	[80,]	0
[15,]	0	[48,]	0	[81,]	0
[16,]	0	[49,]	0	[82,]	0
[17,]	0	[50,]	0	[83,]	1
[18,]	0	[51,]	0	[84,]	0
[19,]	0	[52,]	0	[85,]	0
[20,]	0	[53,]	0	[86,]	0
[21,]	0	[54,]	1	[87,]	0
[22,]	0	[55,]	0	[88,]	0
[23,]	0	[56,]	0	[89,]	0
[24,]	0	[57,]	0	[90,]	0
[25,]	1	[58,]	0	[91,]	0
[26,]	0	[59,]	0	[92,]	0
[27,]	0	[60,]	1	[93,]	0
[28,]	0	[61,]	1	[94,]	0
[29,]	0	[62,]	0	[95,]	1
[30,]	0	[63,]	0	[96,]	0
[31,]	0	[64,]	1	[97,]	0
[32,]	0	[65,]	0	[98,]	0
[33,]	0	[66,]	0	[99,]	0

results	actual		results	actual		results	actual	
[1,] 0	1	X	[34,] 0	1	X	[67,] 0	0	
[2,] 0	1	X	[35,] 1	1		[68,] 1	0	X
[3,] 1	1		[36,] 0	0		[69,] 0	0	
[4,] 0	0		[37,] 0	0		[70,] 1	1	
[5,] 0	0		[38,] 0	0		[71,] 0	1	X
[6,] 0	1	X	[39,] 0	1	X	[72,] 0	1	X
[7,] 0	0		[40,] 0	0		[73,] 0	1	X
[8,] 0	0		[41,] 0	1	X	[74,] 1	0	X
[9,] 0	0		[42,] 0	0		[75,] 0	0	
[10,] 0	0		[43,] 0	0		[76,] 0	0	
[11,] 0	0		[44,] 0	0		[77,] 0	1	X
[12,] 0	0		[45,] 0	0		[78,] 0	0	
[13,] 0	1	X	[46,] 0	0		[79,] 0	0	
[14,] 0	0		[47,] 1	0	X	[80,] 0	0	
[15,] 0	1	X	[48,] 0	0		[81,] 0	1	X
[16,] 0	1	X	[49,] 0	0		[82,] 0	0	
[17,] 0	1	X	[50,] 0	1	X	[83,] 1	0	X
[18,] 0	1	X	[51,] 0	0		[84,] 0	1	X
[19,] 0	0		[52,] 0	1	X	[85,] 0	0	
[20,] 0	0		[53,] 0	0		[86,] 0	1	X
[21,] 0	0		[54,] 1	1		[87,] 0	0	
[22,] 0	0		[55,] 0	1	X	[88,] 0	0	
[23,] 0	0		[56,] 0	0		[89,] 0	0	
[24,] 0	0		[57,] 0	0		[90,] 0	1	X
[25,] 1	0	X	[58,] 0	1	X	[91,] 0	1	X
[26,] 0	0		[59,] 0	0		[92,] 0	1	X
[27,] 0	1	X	[60,] 1	1		[93,] 0	1	X
[28,] 0	0		[61,] 1	1		[94,] 0	1	X
[29,] 0	0		[62,] 0	0		[95,] 1	0	X
[30,] 0	0		[63,] 0	0		[96,] 0	0	
[31,] 0	0		[64,] 1	1		[97,] 0	0	
[32,] 0	0		[65,] 0	0		[98,] 0	0	
[33,] 0	0		[66,] 0	0		[99,] 0	0	

Here I compared the predicted results with the actual results on excel. The neural network managed to get 65/99 correct (65.66%).

Conclusion/Observations

I think the neural network did an okay job at predicting the student's admission based on GRE, GPA, and school rank. The downside of converting the probabilities into binary form is that if there is a probability just under 50%, it will count as a fail so probabilities like 48% could possibly have ended up passing. For instance, #91 has a 48% probability so it converted it to 0 when the actual result was 1 (pass). When looking back at the dataset, I noticed that in the training dataset, 140/200 students failed to be admitted so next time, I would like to try even out the data between those who pass and those who failed for a more even learning process.

R Markdown CODE

```
library(neuralnet)

library(readxl)
```

#Training Data Set

```
train <- read_excel("big_training_data.xlsx")
```

#Normalizing my data points

```
train$gre <- (train$gre - min(train$gre))/(max(train$gre) - min(train$gre))

train$gpa <- (train$gpa - min(train$gpa))/(max(train$gpa) - min(train$gpa))

train$rank <- (train$rank - min(train$rank))/(max(train$rank) - min(train$ran
k))
```

#Neural Network

```
network <- neuralnet(admit ~ ., data = train, hidden = 2, err.fct = "sse", ac
t.fct = "logistic", linear.output = FALSE, rep = 5)

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

#Test Data Set

```
test <- read_excel("big_test_data.xlsx")
```

#Normalizing Test Data Set

```
test$gre <- (test$gre - min(test$gre))/(max(test$gre) - min(test$gre))

test$gpa <- (test$gpa - min(test$gpa))/(max(test$gpa) - min(test$gpa))

test$rank <- (test$rank - min(test$rank))/(max(test$rank) - min(test$rank))
```

#Prediction

```
Predict = compute(network,test)

Predict$net.result
```

#Convert probability to binary

```
prob <- Predict$net.result
pred <- ifelse(prob > 0.5, 1, 0)

pred
```

Resources

- Zhang, Zhongheng. “Neural Networks: Further Insights into Error Function, Generalized Weights and Others.” *Annals of Translational Medicine,* AME Publishing Company, Aug. 2016, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5009026/.
- *Package ‘Neuralnet’ - the Comprehensive R Archive Network*. https://cran.r-project.org/web//packages/neuralnet/neuralnet.pdf.
- “Backpropagation.” *Brilliant Math & Science Wiki*, https://brilliant.org/wiki/backpropagation/#:~:text=Backpropagation%2C%20short%20for%20%22backward%20propagation,to%20the%20neural%20network's%20weights.
- Ravi. “Graduate School Admission Data.” *Kaggle*, 15 Nov. 2017, https://www.kaggle.com/datasets/malapatiravi/graduate-school-admission-data?select=binary1.csv.