Neural Networks For Binary Classification

"Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions"

– Ian Goodfellow, et al[1]

Introduction

If we discuss Neural Networks (NN), we should first consider the system we hope to emulate. Let us start with a simple count of neuronal cells in various organisms along the earth's phylogenetic tree. We might get a better idea of the type of "computing power" these living creatures possess. See table 5.1.

Table 5.1: Organisms Vs Number of Neurons In Each (Wikipedia)

Organism	Common Name	Approximate Number of Neurons
C. elegans	roundworm	302
Chrysaora fuscescens	jellyfish	5,600
Apis linnaeus	honey bee	960,000
Mus musculus	mouse	71,000,000
Felis silvestris	cat	760,000,000
Canis lupus familiaris	dog	2,300,000,000
Homo sapien sapien	humans	100,000,000,000

This table portrays a high-level overview of the computing power of neuronal clusters and brains produced throughout evolution. However, there is one missing number worth noting. The table above does not describe the connectivity between neurons. The connectivity of neurons varies greatly from lower to higher organisms. For example, some simple animals have only "four to eight separate branches," [2] per nerve cell. While human neurons may have approximately 10^4 inter-connected synaptic junctions per neuron, thus resulting in a total of approximately 600 x 10^{12} synapses per human brain. [3]

Although neurons have differing morphologies, neurons in the human brain are extremely diverse. Indeed, size and shape may not be the definitive way of classifying neurons but instead by what neurotransmitters the cells secrete. "Neurotransmitters can be classified as either excitatory or inhibitory."[4] Currently the NeuroPep database "holds 5949 non-redundant neuropeptide entries originating from 493 organisms belonging to 65 neuropeptide families."[5]

[1]Ian Goodfellow, Yoshua Bengio, Aaron Courville, 'Deep Learning', MIT Press, 2016, http://www.deeplearningbook.org
[2]https://www.wormatlas.org/hermaphrodite/nervous/Neuroframeset.html
[3]Shepherd, G. M. (2004), The synaptic organization of the brain (5th ed.), Oxford University Press, New York.
[4]https://www.kenhub.com/en/library/anatomy/neurotransmitters
[5]http://isyslab.info/NeuroPep/home.jsp

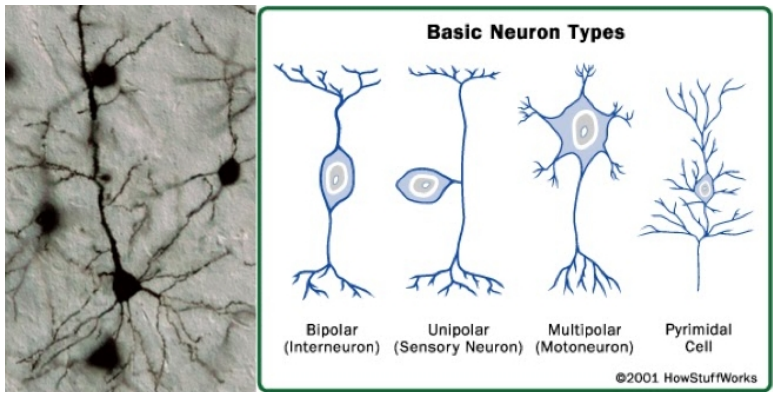


Figure 1: Basic Neuron Types and S.E.M. Image

[6]

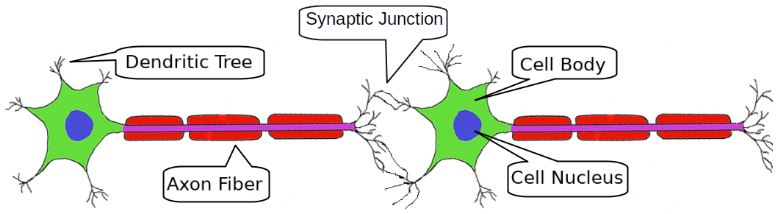


Figure 2: Two Neuron System (Image From The Public Domain)

Given an order of operation via:

Dendrite(s) $\Longrightarrow$ Cell body $\Longrightarrow$ Fibrous Axon $\Longrightarrow$ Synaptic Junction or Synaptic Gap $\Longrightarrow$ Dendrite(s) . . . Ad infinitum.

However, nature is more subtle and intricate than to have neurons in a series, only blinking on and off, firing or not. NN are often programmed to classify dangerous road objects, as is the case of Tesla cars. The goal of a Tesla auto-piloted car is to use all available sensors to correctly classify all the conceivable circumstances on the road. On the road, a Tesla automobile uses dozens of senors which the computer needs to evaluate and weigh the values of all these sensors to formulate a 'decision.' The altitude of the auto, derived from the GPS, may weigh less heavily than the speed of the vehicle or Lidar estimates on how close objects are. However, our goal of safe driving can be thwarted when an artificial intelligence system decides a truck is a sign and does not apply the brakes.[7]

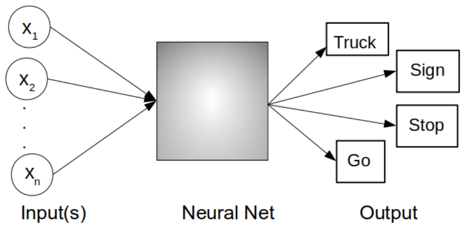


Figure 3: Goal of a Tesla Neural Networks is to generate the correct repsonses for its environment.

[6]https://www.howstuffworks.com/
[7]https://arstechnica.com/cars/2019/05/feds-autopilot-was-active-during-deadly-march-tesla-crash/

The One Neuron System

If we investigate a one neuron system, *our* neuron could be diagrammed in four sections.[8]

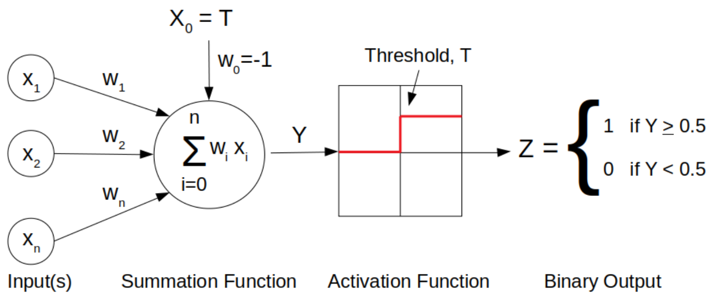


Figure 4: One Neuron Schema

If we investigate one neuron for a moment, we find two separate mathematical functions are being carried out by a single nerve cell.

Summation Function

The first segment is a summation function. It receives the real number values from, x_1 to x_n, all the branches of the dendritic trees, and multiplies them by a set of weights. These X inputs are multiplied by a set of corresponding unique weights from w_1 to w_n. An analogy I prefer is of small or large rivers joining giving a total current. The current moves through the branches giving a total signal or current of sodium ions. Interestingly the summation in each neuron, while dealing with the vectors of inputs and weights, is carrying out the dot product of these vectors, such that;

Initially, the NN used the Heaviside-Threshold Function, as shown in figure 4, the 'One Neuron System.' The benefits of step functions were their simplicity and high signal to noise ratio. While the detriments were, it is a discontinuous function, therefore not able to be differentiated and a mathematical problem.

Let us take into account the product, $x_0 \cdot w_0$. If we assign $x_0 = T$ and $w_0 = -1$ this simply becomes a bias. This bias allows us the ability to shift our Activation Function and its inflection point in the positive or negative x-direction.

$$\hat{Y} \;=\; X^T \cdot W - Bias \;\;\equiv\;\; \sum_{i=0}^{n} x_i w_i - T \tag{1}$$

Activation Functions

The second function is called an Activation Function. Once the Summation Function yields a value, its result is sent to the *Activation Function* or *Threshold Function.*

$$Z^{(1)} = f\left(\sum_{i=0}^{n} x_i w_i - T\right) = \{0, 1\} \tag{2}$$

The function displayed in figure #4, One Neuron Schema, is a step function. However this step function has a problem mathematically, namely it is a discontinuous and therefore not differentiable. This fact is important.

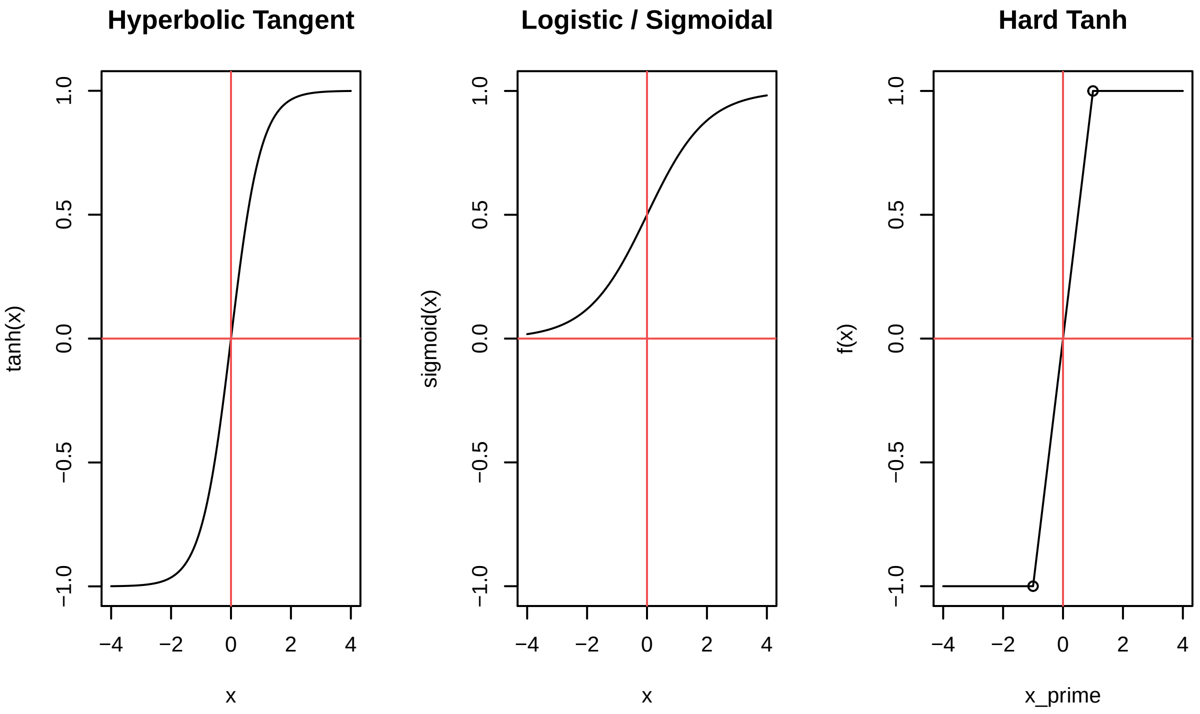
[8]Tom Mitchell, Machine Learning, McGraw-Hill, 1997, ISBN: 0070428077

Therefore several functions may be used in place of the step function. One is the hyperbolic tangent (*tanh*) function, the *sigmoidal* function, a *Hard Tanh*, a *reLU*, and *Softmax* Functions. These have certain advantages, namely they simplify the hyperbolic tangent function. Not only does the Hard Tanh and reLU simplify calculations it is useful for increasing the gain near the asymptotic limits of the sigmoidal and tanh functions. The derivatives of the sigmoidal and tanh functions are very small, near 0 and 1, while the reLU and Hard Tanh slopes are one or zero.

$$Z^{(2)} = tanh(x) = \frac{1-e^{-\alpha}}{1+e^{-\alpha}} \quad : \quad where \quad \alpha = \sum_{i=1}^{n} x_i w_i - T \tag{3}$$

$$Z^{(3)} = sigmoid(x) = \frac{1}{1+e^{-\alpha}} \tag{4}$$

$$Z^{(4)} = Hard\ Tanh(x) = \begin{cases} 1 & x > 1 \\ x & -1 \leq x \leq 1 \\ -1 & x < -1 \end{cases} \tag{5}$$



Several alternative functions are useful for various reasons. The most common of which are Softmax and reLU functions.

Rectified Linear Activation Unit, (ReLU):

$$Z^{(5)} = ReLU = \begin{cases} x \geqq 0 & y = x \\ x < 0 & y = 0 \end{cases} \tag{6}$$

Binary Output Or Probability

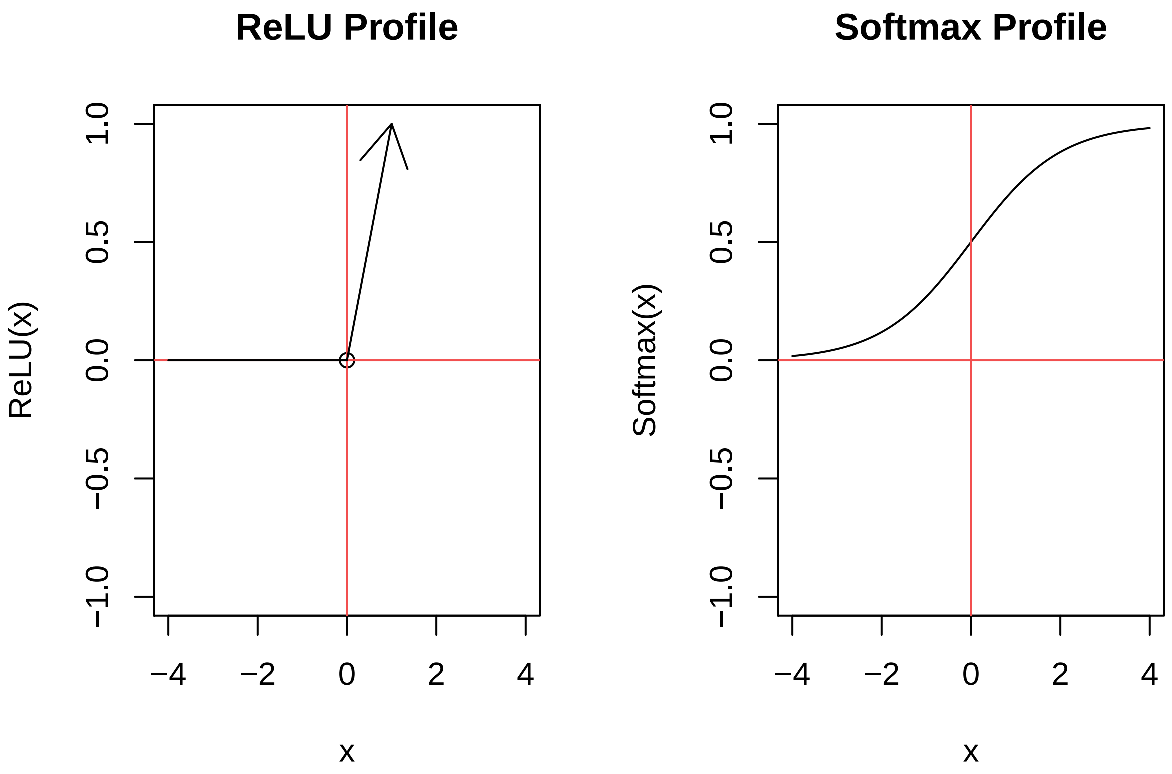
In the case of real neurons, the output is off or on, zero or one. However, in the case of our electronic model, it is advantageous to calculate a probability for greater interpretability.

The Softmax function may appear like the Sigmoid function from above but it differs in major ways.[9]

- The softmax activation function returns the probability distribution over mutually exclusive output classes.
- The calculated probabilities will be in the range of 0 to 1.
- The sum of all the probabilities is equals to 1.

Typically the Softmax Function is used in binary or multiple classification logistic regression models and in building the final output layer of NN.

$$Z^{(6)} \; = \; Softmax(x) = \frac{e^{\alpha_i}}{\sum_{i=1}^{n} e^{\alpha_i}} \tag{7}$$



The benefit of these activation functions is that they are now differentiable. This fact becomes important for *Back-Propagation*, which is discussed later.

[9]Josh Patterson, Adam Gibson, Deep Learning; A Practitioner's Approach, 2017, O'Rreilly

The Two Neuron System

Building up in complexity, let us could consider our first Neural Network by using *only* two neurons. In two neuron systems, let us first generalize a bit more by adding that X is an array of all the inputs as is W_1 and W_2 is also an array of weights for each neuron. See figure #5.

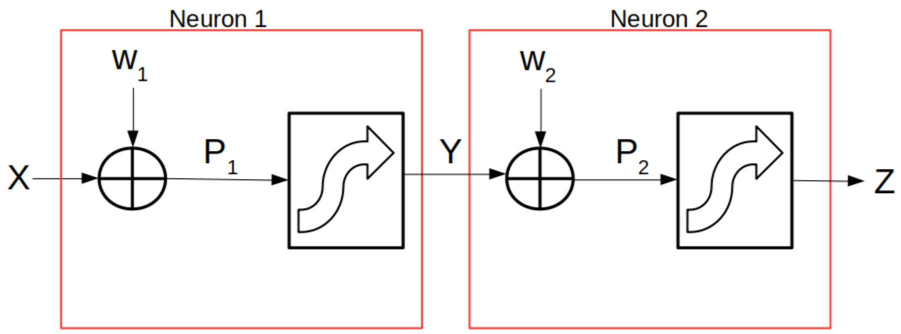


Figure 5: A Two Neuron System

Feed-Forward In A Two Neuron Network

In our two neuron network, we can now write out the mathematics for each step as it progresses in a "forward" (left to right) direction.

Step #1: To move from X to P_1

$$f^1(\overrightarrow{x},\overrightarrow{w}) \equiv \quad P_1 = \left(X^T \cdot W_1 - T\right) \tag{8}$$

Step #2: P_1 feeds forward to Y

$$f^2(P_1) \quad \equiv \hat{Y} = \left(\frac{1}{1+e^{-\alpha}}\right) \quad : \quad where \quad \alpha = P_1 \tag{9}$$

Step #3: Y feeds forward to P_2

$$f^3(\overrightarrow{y},\overrightarrow{w}) \quad \equiv \; P_2 = \left(Y^T \cdot W_2 - T\right) \tag{10}$$

Step #4: P_2 feeds forward to Z

$$f^4(P_2) \quad \equiv \hat{Z} = \left(\frac{1}{1+e^{-\alpha}}\right) \quad : \quad where \;\; \alpha = P_2 \tag{11}$$

Step #5: Our complicated function is simply a matter of chaining one result so that it may be used in the next step.

$$\hat{Z} \;=\; f^4\left(f^3\left(f^2\left(f^1\left(X,W\right)\right)\right)\right) \tag{12}$$

In our **Feed-Forward Propagation**, we can now take the values from any numerical system and produce zeros, ones, or probabilities. Remember, in this set of experiments, we are using the concentrations of the 20 amino acids to provide a categorical or binary output, belongs to Myoglobin protein family, or does not.

Error Back-propagation

Now that we have learned to calculate the output of our neurons using the Feed-Forward process, what if our final answer is incorrect? Can we build a feed back system to determine the weights needed to obtain our desired value of $\hat{z}$? The answer is yes. The process for determining the weights is known as Back-Propagation. Back-Propagation, also known as error back-propagation, is crucial to understanding and tuning a neural network.

Simply stated Back-Propagation is an optimization routine which iteratively calculates the errors that occur at each stage of a neural network. Back-Propagation uses the partial derivatives of the feed forward functions, specifically. The chain rule and gradient descent are also used to determine the weights ($W_1 \ \ and \ \ W_2$) which are propagated through the network to find weights used in the summation step of a neuron.[10]

This thumbnail sketch gives the building blocks to calculate W which can be run until we reach a value that we desire. However the first time the back-propagation is carried out all the weights are chosen randomly. If the weights were set to the same number there would be no change throughout the system.

In the two neuron system, our first step is to generate an error or performance (Perf) function to minimize. If we call d our desired value, we can minimize the square error, a common choice.[11]

Step #1: Performance (Perf)

$$\mathbf{Perf} \quad = \quad c \cdot (d - \hat{z})^2 \tag{13}$$

Step #2:

$$\frac{dZ}{dx} \quad = \quad \frac{d\left\{f^4\left(f^3\left(f^2\left(f^1\left(X, W\right)\right)\right)\right)\right\}}{dx} \tag{14}$$

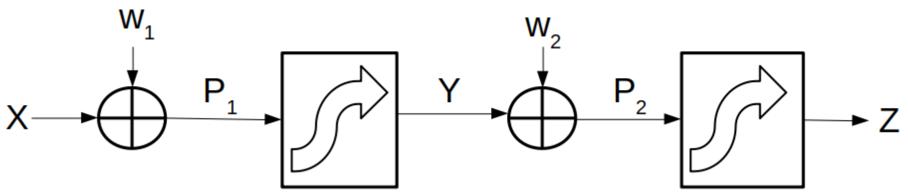


Figure 6: A Two Neuron System

Using the chain-rule, working backward and the 'Two Neuron System' figure as a guide through the error back-propagation, we find:

Step #3: Neuron 2 $\Rightarrow$ 1

$$\frac{\delta Perf}{\delta w_1} \quad = \quad \frac{\delta Perf}{\delta z} \cdot \frac{\delta z}{\delta P_2} \cdot \frac{\delta P_2}{\delta y} \cdot \frac{\delta y}{\delta P_1} \cdot \frac{\delta P_2}{\delta w_1} \tag{15}$$

Step #4: Performance

$$\frac{\delta Perf}{\delta z} \quad = \quad \frac{\delta\left\{\frac{1}{2}\|\overrightarrow{d} - \overrightarrow{z}\|^2\right\}}{\delta z} \quad = \quad \overrightarrow{d} - \overrightarrow{z} \tag{16}$$

Step #5: Substitute $P_2 = \alpha$

$$\frac{\delta z}{\delta P_2} \quad = \quad \frac{\delta\ \left((1 + e^{-\alpha})^{-1}\right)}{\delta \alpha} \quad = \quad e^{-\alpha} \cdot (1 + e^{-\alpha})^{-2} \tag{17}$$

[10] David Rumelhart, Geoffrey Hinton, & Ronald Williams, Learning represetnations by back-propagating Errors, Nature, 323, 533-536, Oct. 9, 1986

[11] Ivan N. da Silva, Danilo H. Spatti, Rogerio A. Flauzino, Luisa H. B. Liboni, Silas F. dos Reis Alves, Artificial Neural Networks: A Practical Course, DOI 10.1007/978-3-319-43162-8, 2017

Step #6: Rearrange the expression

$$\frac{e^{-\alpha}}{(1+e^{-\alpha})^{-2}} \quad = \quad \frac{e^{-\alpha}}{1+e^{-\alpha}} \cdot \frac{1}{1+e^{-\alpha}} \tag{18}$$

Step #7: Add 1 *and* subtract 1

$$= \quad \frac{(1+e^{-\alpha})-1}{1+e^{-\alpha}} \cdot \frac{1}{1+e^{-\alpha}} \tag{19}$$

Step #8: Rearrange to find

$$= \quad \left(\frac{1+e^{-\alpha}}{1+e^{-\alpha}} - \frac{1}{1+e^{-\alpha}}\right)\left(\frac{1}{1+e^{-\alpha}}\right) \quad = \quad \left(1 - \frac{1}{1+e^{-\alpha}}\right)\left(\frac{1}{1+e^{-\alpha}}\right) \tag{20}$$

Step #9: Therefore we find

$$\frac{\delta z}{\delta \alpha} \quad = \quad \frac{\delta\left((1+e^{-\alpha})^{-1}\right)}{\delta \alpha} \quad = \quad \left(1 - \frac{1}{1+e^{-\alpha}}\right)\left(\frac{1}{1+e^{-\alpha}}\right) \tag{21}$$

Nevertheless, we need one more part to ascertain the weights. As the error back-propagation is computed this process does not reveal how much the weights need to be adjusted/changed to compute the next round of weights given their current errors. For this we require one last equation or concept.

Once we compute the weights from our chain rule set of equations we must change the values in the direction proportional to the change in error. This is performed by using gradient descent.

Step #10: Learning Rate

$$\Delta W \ : \ W_{i+1} \ = \ W_i \ - \ \eta \cdot \frac{\delta Perf}{\delta W} \tag{22}$$

where η is the learning rate for the system. The key to the learning rate is that it must be sought and its range mapped for optimum efficiency. However smaller rates have the advantage of not overshooting the desired minimum/maximum. If the learning rate is too large the values of W may jump wildly and not settle into a max/min. There is a fine balance that must be considered such that the weights are not trapped in a local minimum and wildly oscillate unable to converge.

The last step of *error back-propagation* is simply setting up the derivatives mechanically and is not shown for brevity.

Neural Network Experiment For Binary Classfication

```r
# Load Libraries
Libraries <- c("dplyr", "knitr", "readr", "caret", "MASS", "nnet", "purrr", "doMC")
for (p in Libraries) {
    library(p, character.only = TRUE)
}
```

```r
education <- read_csv("../00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv",
                      col_types = cols(Class = col_factor(levels = c("0","1")),
                                       PID = col_skip(),
                                       TotalAA = col_skip()))
```

Create Training Data

```r
set.seed(1000)
# Stratified sampling
TrainingDataIndex <- createDataPartition(education$Class, p = 0.8, list = FALSE)

# Create Training Data
trainingData <- education[ TrainingDataIndex, ]
testData     <- education[-TrainingDataIndex, ]

TrainingParameters <- trainControl(method = "repeatedcv",
                                   number = 5,
                                   repeats = 5,
                                   savePredictions = "final") # IMPORTANT: Saves predictions
```

Train model with neural networks

```r
end_time - start_time            # Display time
```

```
## Time difference of 20.28306 secs
```

Confusion Matrix and Statistics

```r
NNPredictions <- predict(NNModel, testData)

# Create confusion matrix
cmNN <-confusionMatrix(NNPredictions, testData$Class)
print(cmNN)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 237  12
##          1   6 212
##
##                Accuracy : 0.9615
##                  95% CI : (0.9398, 0.977)
##     No Information Rate : 0.5203
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9227
##
##  Mcnemar's Test P-Value : 0.2386
##
##             Sensitivity : 0.9753
##             Specificity : 0.9464
##          Pos Pred Value : 0.9518
##          Neg Pred Value : 0.9725
##              Prevalence : 0.5203
##          Detection Rate : 0.5075
##    Detection Prevalence : 0.5332
##       Balanced Accuracy : 0.9609
##
```

```
##        'Positive' Class : 0
##
```

```
NNModel
```

```
## Neural Network
##
## 1873 samples
##   20 predictor
##    2 classes: '0', '1'
##
## Pre-processing: scaled (20), centered (20)
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 1498, 1498, 1499, 1499, 1498, 1499, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.9339109  0.8671305
##   1     1e-04  0.9346544  0.8686793
##   1     1e-01  0.9481084  0.8958797
##   3     0e+00  0.9565473  0.9128846
##   3     1e-04  0.9564355  0.9126691
##   3     1e-01  0.9607087  0.9212480
##   5     0e+00  0.9571825  0.9142100
##   5     1e-04  0.9570772  0.9139950
##   5     1e-01  0.9625266  0.9249177
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 5 and decay = 0.1.
```

```
## Obtain List of False Positives & False Negatives
fp_fn_NNModel <- NNModel %>% pluck("pred") %>% dplyr::filter(obs != pred)

# Write CSV in R
write.table(fp_fn_NNModel,
            file = "../00-data/03-ml_results/fp_fn_NN.csv",
            row.names = FALSE,
            na = "",
            col.names = TRUE,
            sep = ",")

nrow(fp_fn_NNModel) ## NOTE: NOT UNIQUE NOR SORTED
```

```
## [1] 351
```

False Positive & False Negative Neural Network set

```
keep <- "rowIndex"

fp_fn_NN <- read_csv("../00-data/03-ml_results/fp_fn_NN.csv")

NN_fp_fn_nums <- sort(unique(unlist(fp_fn_NN[, keep], use.names = FALSE)))

length(NN_fp_fn_nums)
```

```
## [1] 140
```

```
NN_fp_fn_nums
```

```
##   [1]    1    2    4    6   10   15   46   57   58   88  100  114  115  116  130
##  [16]  136  141  146  150  170  172  182  183  185  191  201  231  239  249  254
##  [31]  260  349  407  421  427  430  436  439  449  452  453  503  516  518  526
##  [46]  530  531  532  534  536  537  542  546  547  551  554  562  566  570  573
##  [61]  575  576  577  580  583  584  589  592  655  910  913  980 1032 1033 1034
##  [76] 1035 1041 1067 1069 1092 1093 1094 1096 1100 1101 1106 1115 1121 1130 1144
##  [91] 1190 1219 1222 1223 1226 1233 1234 1245 1264 1279 1282 1340 1471 1482 1484
## [106] 1487 1510 1522 1569 1571 1575 1576 1577 1579 1582 1585 1587 1588 1589 1594
## [121] 1600 1608 1618 1619 1621 1622 1623 1693 1723 1734 1771 1780 1789 1828 1829
## [136] 1830 1831 1832 1833 1873
```

```
write_csv(x = as.data.frame(NN_fp_fn_nums),
          path = "../00-data/04-sort_unique_outliers/NN_nums.csv")
```

- The 'NN' set included a total of 140 unique observations containing both FP and FN.

Neural Network Conclusion

Lorem