# CPTR330 – Homework 5

Pierre Visconti

May 1, 2023

## ANN Algorithm

Artificial Neural Networks (ANN) use a model derived from our biological understanding of how our brains work to model the relationships between a set of input signals and an output signal. Each input is weighted by its determined importance and the net signals from all the inputs are fed to an activator function which determines whether the input threshold has been attained. Typically the weights will be adjusted with each iteration with the goal of minimizing error, the iterations are referred to as epochs. Strengths of ANN are that it can be adapted for both classification or numeric predictions, and it makes very little assumptions about the data. Some weaknesses are that it is extremely computationally intensive, and prone to overfitting.

### Step 1 - Collect Data

The dataset is called "Johns Hopkins University Ionosphere database". The source is the Space Physics Group at the Applied Physics Labratory at Johns Hopkins University. The dataset contains radar data that was collected by a system of 16 high-frequency antennas targetting free electrons in the ionosphere. The signals received by the antennas were processed and split into two seperate features for all 17 pulse numbers.

### Step 2 - Exploring And Preparing The Data

Import the data

```
data = read.csv("ionosphere.data", header=FALSE)
```

Looking over the data

```
str(data)
```

```
## 'data.frame':    351 obs. of  35 variables:
##  $ V1 : int  1 1 1 1 1 1 1 0 1 1 ...
##  $ V2 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V3 : num  0.995 1 1 1 1 ...
##  $ V4 : num  -0.0589 -0.1883 -0.0336 -0.4516 -0.024 ...
##  $ V5 : num  0.852 0.93 1 1 0.941 ...
##  $ V6 : num  0.02306 -0.36156 0.00485 1 0.06531 ...
##  $ V7 : num  0.834 -0.109 1 0.712 0.921 ...
##  $ V8 : num  -0.377 -0.936 -0.121 -1 -0.233 ...
##  $ V9 : num  1 1 0.89 0 0.772 ...
##  $ V10: num  0.0376 -0.0455 0.012 0 -0.164 ...
##  $ V11: num  0.852 0.509 0.731 0 0.528 ...
##  $ V12: num  -0.1776 -0.6774 0.0535 0 -0.2028 ...
##  $ V13: num  0.598 0.344 0.854 0 0.564 ...
##  $ V14: num  -0.44945 -0.69707 0.00827 0 -0.00712 ...
##  $ V15: num  0.605 -0.517 0.546 -1 0.344 ...
##  $ V16: num  -0.38223 -0.97515 0.00299 0.14516 -0.27457 ...
```

```
##  $ V17: num  0.844 0.055 0.838 0.541 0.529 ...
##  $ V18: num  -0.385 -0.622 -0.136 -0.393 -0.218 ...
##  $ V19: num  0.582 0.331 0.755 -1 0.451 ...
##  $ V20: num  -0.3219 -1 -0.0854 -0.5447 -0.1781 ...
##  $ V21: num  0.5697 -0.1315 0.7089 -0.6997 0.0598 ...
##  $ V22: num  -0.297 -0.453 -0.275 1 -0.356 ...
##  $ V23: num  0.3695 -0.1806 0.4339 0 0.0231 ...
##  $ V24: num  -0.474 -0.357 -0.121 0 -0.529 ...
##  $ V25: num  0.5681 -0.2033 0.5753 1 0.0329 ...
##  $ V26: num  -0.512 -0.266 -0.402 0.907 -0.652 ...
##  $ V27: num  0.411 -0.205 0.59 0.516 0.133 ...
##  $ V28: num  -0.462 -0.184 -0.221 1 -0.532 ...
##  $ V29: num  0.2127 -0.1904 0.431 1 0.0243 ...
##  $ V30: num  -0.341 -0.116 -0.174 -0.201 -0.622 ...
##  $ V31: num  0.4227 -0.1663 0.6044 0.2568 -0.0571 ...
##  $ V32: num  -0.5449 -0.0629 -0.2418 1 -0.5957 ...
##  $ V33: num  0.1864 -0.1374 0.5605 -0.3238 -0.0461 ...
##  $ V34: num  -0.453 -0.0245 -0.3824 1 -0.657 ...
##  $ V35: chr  "g" "b" "g" "b" ...
```

```
summary(data)
```

```
##        V1               V2          V3                V4
##  Min.   :0.0000   Min.   :0   Min.   :-1.0000   Min.   :-1.00000
##  1st Qu.:1.0000   1st Qu.:0   1st Qu.: 0.4721   1st Qu.:-0.06474
##  Median :1.0000   Median :0   Median : 0.8711   Median : 0.01631
##  Mean   :0.8917   Mean   :0   Mean   : 0.6413   Mean   : 0.04437
##  3rd Qu.:1.0000   3rd Qu.:0   3rd Qu.: 1.0000   3rd Qu.: 0.19418
##  Max.   :1.0000   Max.   :0   Max.   : 1.0000   Max.   : 1.00000
##        V5                V6                V7                V8
##  Min.   :-1.0000   Min.   :-1.0000   Min.   :-1.0000   Min.   :-1.00000
##  1st Qu.: 0.4127   1st Qu.:-0.0248   1st Qu.: 0.2113   1st Qu.:-0.05484
##  Median : 0.8092   Median : 0.0228   Median : 0.7287   Median : 0.01471
##  Mean   : 0.6011   Mean   : 0.1159   Mean   : 0.5501   Mean   : 0.11936
##  3rd Qu.: 1.0000   3rd Qu.: 0.3347   3rd Qu.: 0.9692   3rd Qu.: 0.44567
##  Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.00000
##        V9                V10               V11               V12
##  Min.   :-1.00000   Min.   :-1.00000   Min.   :-1.00000   Min.   :-1.00000
##  1st Qu.: 0.08711   1st Qu.:-0.04807   1st Qu.: 0.02112   1st Qu.:-0.06527
##  Median : 0.68421   Median : 0.01829   Median : 0.66798   Median : 0.02825
##  Mean   : 0.51185   Mean   : 0.18135   Mean   : 0.47618   Mean   : 0.15504
##  3rd Qu.: 0.95324   3rd Qu.: 0.53419   3rd Qu.: 0.95790   3rd Qu.: 0.48237
##  Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000
##       V13               V14               V15               V16
##  Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.00000
##  1st Qu.: 0.0000   1st Qu.:-0.07372   1st Qu.: 0.0000   1st Qu.:-0.08170
##  Median : 0.6441   Median : 0.03027   Median : 0.6019   Median : 0.00000
##  Mean   : 0.4008   Mean   : 0.09341   Mean   : 0.3442   Mean   : 0.07113
##  3rd Qu.: 0.9555   3rd Qu.: 0.37486   3rd Qu.: 0.9193   3rd Qu.: 0.30897
##  Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000
##       V17               V18                V19               V20
##  Min.   :-1.0000   Min.   :-1.000000   Min.   :-1.0000   Min.   :-1.00000
##  1st Qu.: 0.0000   1st Qu.:-0.225690   1st Qu.: 0.0000   1st Qu.:-0.23467
##  Median : 0.5909   Median : 0.000000   Median : 0.5762   Median : 0.00000
##  Mean   : 0.3819   Mean   :-0.003617   Mean   : 0.3594   Mean   :-0.02402
```

```
##  3rd Qu.: 0.9357   3rd Qu.: 0.195285   3rd Qu.: 0.8993   3rd Qu.: 0.13437
##  Max.   : 1.0000   Max.   : 1.000000   Max.   : 1.0000   Max.   : 1.00000
##       V21               V22                V23               V24
##  Min.   :-1.0000   Min.   :-1.000000   Min.   :-1.0000   Min.   :-1.00000
##  1st Qu.: 0.0000   1st Qu.:-0.243870   1st Qu.: 0.0000   1st Qu.:-0.36689
##  Median : 0.4991   Median : 0.000000   Median : 0.5318   Median : 0.00000
##  Mean   : 0.3367   Mean   : 0.008296   Mean   : 0.3625   Mean   :-0.05741
##  3rd Qu.: 0.8949   3rd Qu.: 0.188760   3rd Qu.: 0.9112   3rd Qu.: 0.16463
##  Max.   : 1.0000   Max.   : 1.000000   Max.   : 1.0000   Max.   : 1.00000
##       V25               V26                V27               V28
##  Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.00000
##  1st Qu.: 0.0000   1st Qu.:-0.33239   1st Qu.: 0.2864   1st Qu.:-0.44316
##  Median : 0.5539   Median :-0.01505   Median : 0.7082   Median :-0.01769
##  Mean   : 0.3961   Mean   :-0.07119   Mean   : 0.5416   Mean   :-0.06954
##  3rd Qu.: 0.9052   3rd Qu.: 0.15676   3rd Qu.: 0.9999   3rd Qu.: 0.15354
##  Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000
##       V29               V30                V31               V32
##  Min.   :-1.0000   Min.   :-1.00000   Min.   :-1.0000   Min.   :-1.000000
##  1st Qu.: 0.0000   1st Qu.:-0.23689   1st Qu.: 0.0000   1st Qu.:-0.242595
##  Median : 0.4966   Median : 0.00000   Median : 0.4428   Median : 0.000000
##  Mean   : 0.3784   Mean   :-0.02791   Mean   : 0.3525   Mean   :-0.003794
##  3rd Qu.: 0.8835   3rd Qu.: 0.15407   3rd Qu.: 0.8576   3rd Qu.: 0.200120
##  Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.000000
##       V33               V34               V35
##  Min.   :-1.0000   Min.   :-1.00000   Length:351
##  1st Qu.: 0.0000   1st Qu.:-0.16535   Class :character
##  Median : 0.4096   Median : 0.00000   Mode  :character
##  Mean   : 0.3494   Mean   : 0.01448
##  3rd Qu.: 0.8138   3rd Qu.: 0.17166
##  Max.   : 1.0000   Max.   : 1.00000
```

The dataset contains 35 features and 351 observations. All of the features are continous variables except for the label feature which is of character type with 'b' for bad and 'g' for good.

The V2 feature is uniquely 0 so we can remove it. The rest of the features are already normalized between -1 and 1, except for V1 which is 0-1, so we do not need to normalize the dataset. The label feature also needs to be converted to a binary variable from a character.

```
# removing V2 feature of all 0
data = data[-2]
# changing label feature to numerical values
data$V35[data$V35 == 'b'] = 0
data$V35[data$V35 == 'g'] = 1
# setting V35 to numerical type
data$V35 = as.numeric(data$V35)
```

Checking data to confirm changes

```
str(data)
```

```
## 'data.frame':    351 obs. of  34 variables:
##  $ V1 : int  1 1 1 1 1 1 1 0 1 1 ...
##  $ V3 : num  0.995 1 1 1 1 ...
##  $ V4 : num  -0.0589 -0.1883 -0.0336 -0.4516 -0.024 ...
##  $ V5 : num  0.852 0.93 1 1 0.941 ...
##  $ V6 : num  0.02306 -0.36156 0.00485 1 0.06531 ...
##  $ V7 : num  0.834 -0.109 1 0.712 0.921 ...
```

```
##  $ V8 : num  -0.377 -0.936 -0.121 -1 -0.233 ...
##  $ V9 : num  1 1 0.89 0 0.772 ...
##  $ V10: num  0.0376 -0.0455 0.012 0 -0.164 ...
##  $ V11: num  0.852 0.509 0.731 0 0.528 ...
##  $ V12: num  -0.1776 -0.6774 0.0535 0 -0.2028 ...
##  $ V13: num  0.598 0.344 0.854 0 0.564 ...
##  $ V14: num  -0.44945 -0.69707 0.00827 0 -0.00712 ...
##  $ V15: num  0.605 -0.517 0.546 -1 0.344 ...
##  $ V16: num  -0.38223 -0.97515 0.00299 0.14516 -0.27457 ...
##  $ V17: num  0.844 0.055 0.838 0.541 0.529 ...
##  $ V18: num  -0.385 -0.622 -0.136 -0.393 -0.218 ...
##  $ V19: num  0.582 0.331 0.755 -1 0.451 ...
##  $ V20: num  -0.3219 -1 -0.0854 -0.5447 -0.1781 ...
##  $ V21: num  0.5697 -0.1315 0.7089 -0.6997 0.0598 ...
##  $ V22: num  -0.297 -0.453 -0.275 1 -0.356 ...
##  $ V23: num  0.3695 -0.1806 0.4339 0 0.0231 ...
##  $ V24: num  -0.474 -0.357 -0.121 0 -0.529 ...
##  $ V25: num  0.5681 -0.2033 0.5753 1 0.0329 ...
##  $ V26: num  -0.512 -0.266 -0.402 0.907 -0.652 ...
##  $ V27: num  0.411 -0.205 0.59 0.516 0.133 ...
##  $ V28: num  -0.462 -0.184 -0.221 1 -0.532 ...
##  $ V29: num  0.2127 -0.1904 0.431 1 0.0243 ...
##  $ V30: num  -0.341 -0.116 -0.174 -0.201 -0.622 ...
##  $ V31: num  0.4227 -0.1663 0.6044 0.2568 -0.0571 ...
##  $ V32: num  -0.5449 -0.0629 -0.2418 1 -0.5957 ...
##  $ V33: num  0.1864 -0.1374 0.5605 -0.3238 -0.0461 ...
##  $ V34: num  -0.453 -0.0245 -0.3824 1 -0.657 ...
##  $ V35: num  1 0 1 0 1 0 1 0 1 0 ...
```

create training and testing datasets

```
set.seed(330) # setting seed
# create training and test data by randomly sampling
train.size = round(nrow(data)*0.75) # 75% of the dataset used for training
train.ind = sample(1:nrow(data), train.size)
data_train = data[train.ind,]
data_test = data[-train.ind,]
```

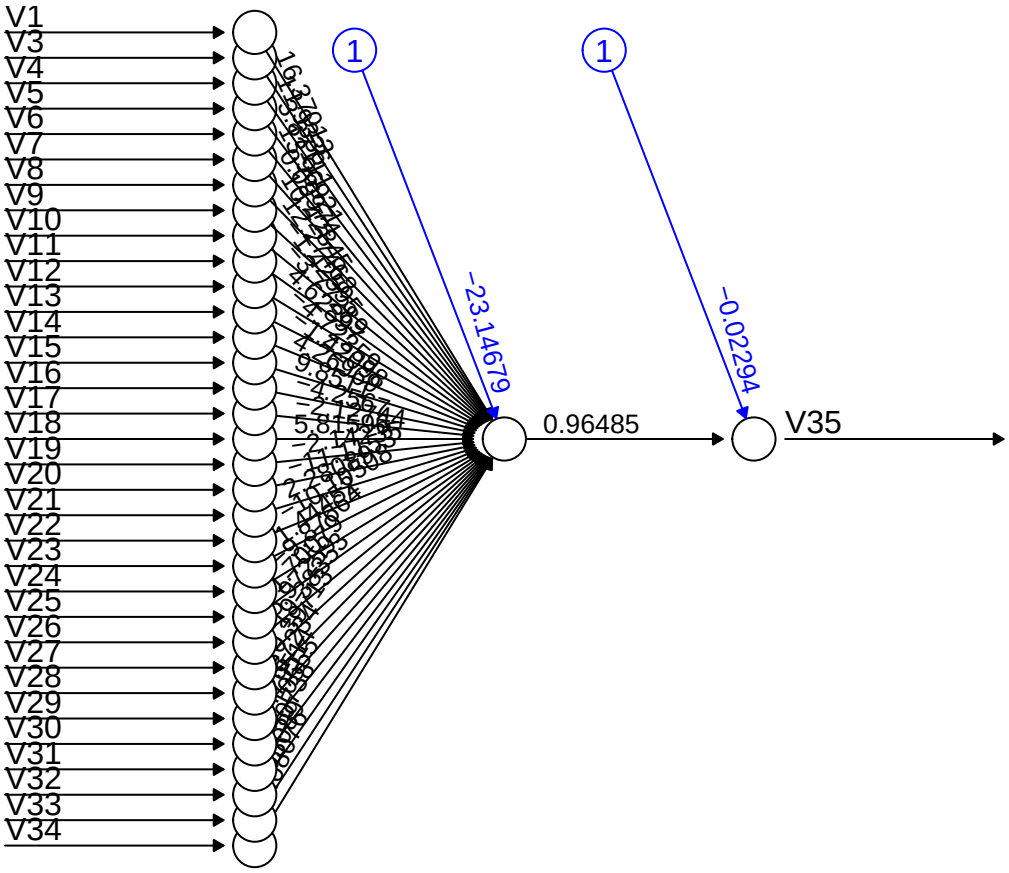
We are now ready to train the model

## Step 3 - Training A Model On The Data

We train the model using the neuralnet function from the neuralnet library. The command requires that we pass our dataset and also specify the class variable (ouput signal) and the features we want to give as input (input signals). In this case we will pass all of the features in our dataset to the model.

training model with default parameters, 1 hidden layer with 1 hidden node.

```
set.seed(330) # setting seed
model = neuralnet(V35~.,data=data_train)
plot(model, rep = "best")
```




## Step 4 - Evaluating Model Performance

Test model performance on both training and testing sets

```
# predict
model_results_train = neuralnet::compute(model, data_train[-34])
model_results_test = neuralnet::compute(model, data_test[-34])
# obtain predicted strength values
predicted_train = model_results_train$net.result
predicted_test = model_results_test$net.result
# examine the correlation between predicted and actual values
print("Training set:")
```

```
## [1] "Training set:"
```

```
cor(predicted_train, data_train[34])
```

```
##            V35
## [1,] 0.9083016
```

```
print("Testing set:")
```

```
## [1] "Testing set:"
```

```
cor(predicted_test, data_test[34])
```

```
##            V35
## [1,] 0.8236753
```

```
print("Training time:")
```

```
## [1] "Training time:"
```

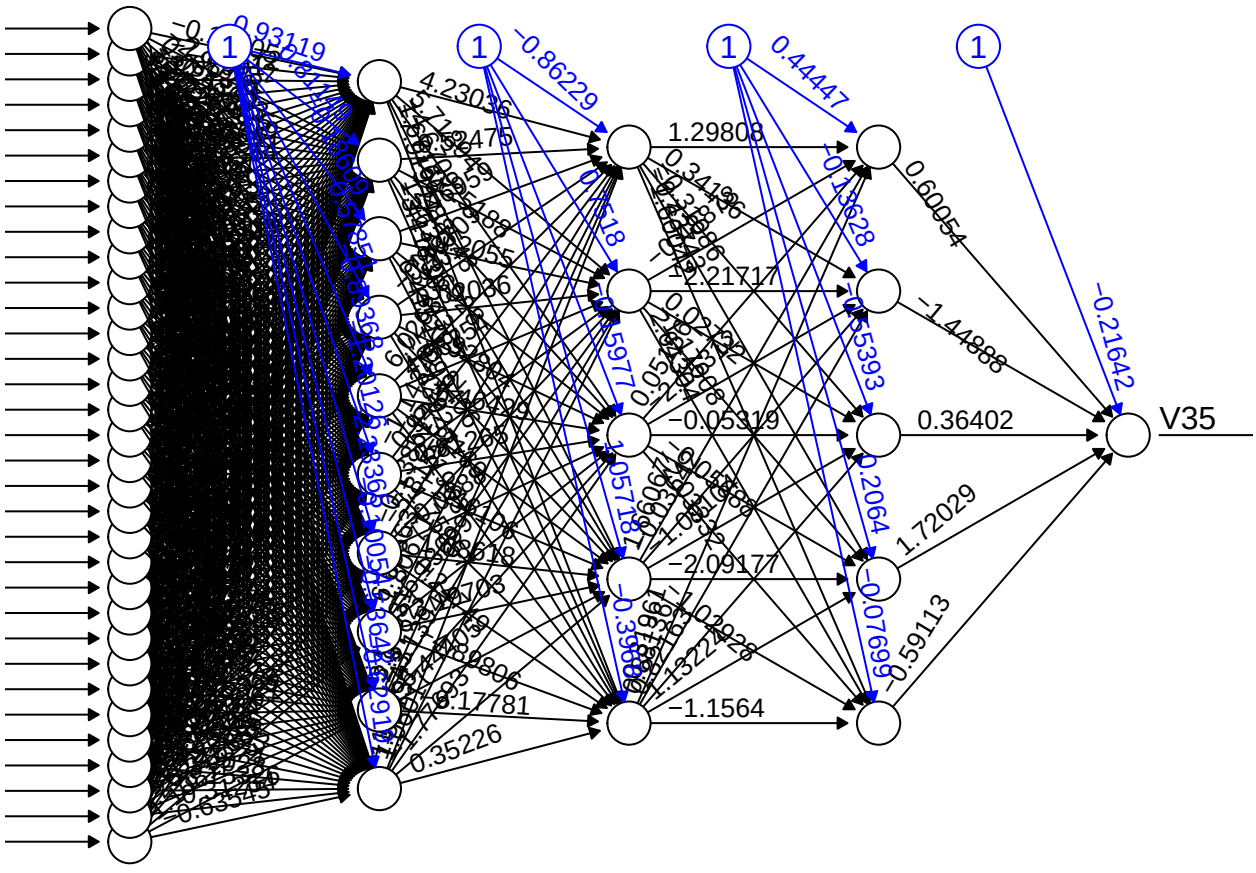
The model achieved an accuracy of 0.823 on the testing set and 0.908 on the training set. The results show us that we are experiencing some overfitting with the model since we achieved significantly better results on the training set, but we still got fairly good results with the testing set so the model is effective.

## Step 5 - Improving Model Performance

We have several options available to improve the performance of the model. We currently only have 1 hidden node with 1 hidden layer, so we could try adding more hidden nodes to our layer and/or adding more hidden layers to create a deep artificial neural network. We could also try using a different activator function.

Training model with multiple hidden nodes and layers. First hidden layer has 10 nodes, the second has 5 nodes, and the third has 5 nodes.

```
set.seed(330)
model = neuralnet(V35~.,data=data_train, hidden=c(10,5,5))
plot(model, rep = "best")
```




Test model performance on both training and testing sets

```
# predict
model_results_train = neuralnet::compute(model, data_train[-34])
model_results_test = neuralnet::compute(model, data_test[-34])
# obtain predicted strength values
predicted_train = model_results_train$net.result
```

```
predicted_test = model_results_test$net.result
# examine the correlation between predicted and actual values
print("Training set:")
```

```
## [1] "Training set:"
```

```
cor(predicted_train, data_train[34])
```

```
##           V35
## [1,] 0.995617
```

```
print("Testing set:")
```

```
## [1] "Testing set:"
```

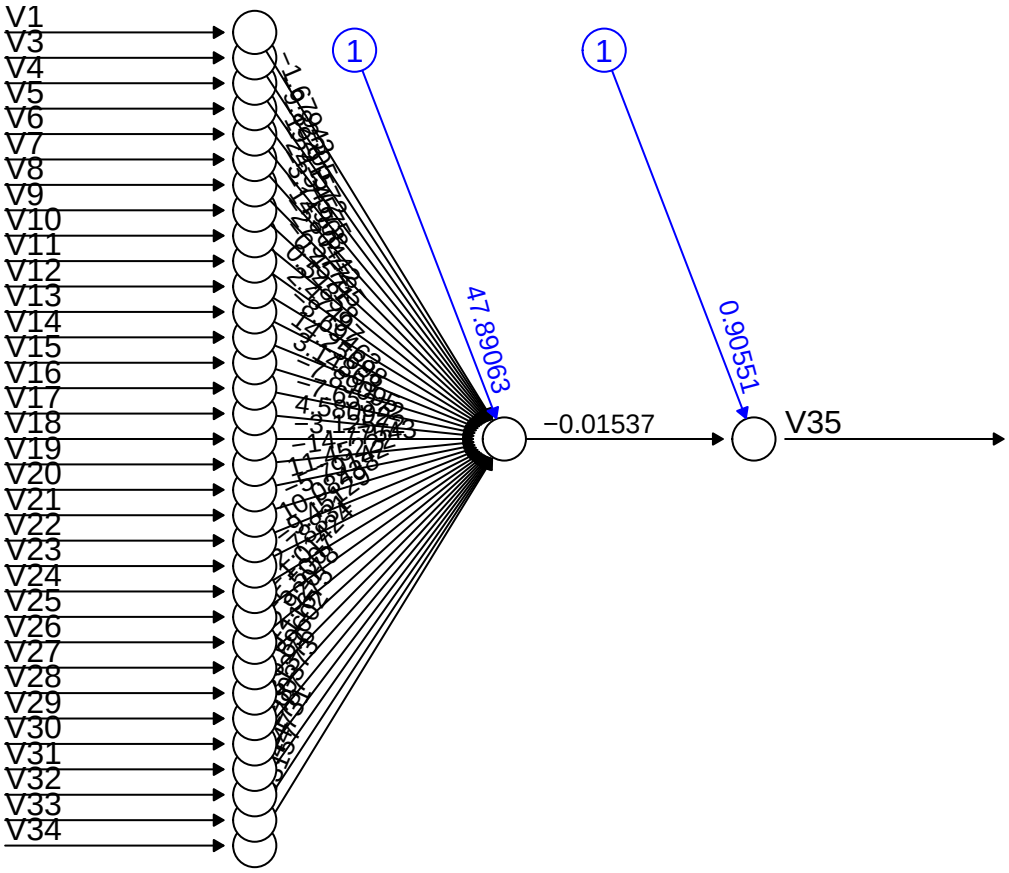
```
cor(predicted_test, data_test[34])
```

```
##            V35
## [1,] 0.9152183
```

This model performed significantly better than our original mode. We achieved an accuracy of 0.915 on the testing set over the 0.823 from before and 0.995 on the training set compared to 0.908 from the original. Again, there is some overfitting occurring, but the model still performed very well on the testing set.

Using a different activator function: using a softplus activator function. To compare results with the original model all other parameters were left the same (1 hidden layer, 1 hidden nodes).

```
set.seed(330) # setting seed
# defining our softplus activator function
softplus <- function(x) { log(1 + exp(x)) }
# training the model
model = neuralnet(V35~.,data=data_train, act.fct=softplus)
plot(model, rep = "best")
```




Test model performance on both training and testing sets

```
# predict
model_results_train = neuralnet::compute(model, data_train[-34])
model_results_test = neuralnet::compute(model, data_test[-34])
# obtain predicted strength values
predicted_train = model_results_train$net.result
predicted_test = model_results_test$net.result
# examine the correlation between predicted and actual values
print("Training set:")
```

```
## [1] "Training set:"
```

```
cor(predicted_train, data_train[34])
```

```
##            V35
## [1,] 0.7688605
```

```
print("Testing set:")
```

```
## [1] "Testing set:"
```

```
cor(predicted_test, data_test[34])
```

```
##           V35
## [1,] 0.754411
```

The model with the softplus activator function performed significantly worst than our previous improved model and the original model. It achieved 0.733 on the testing set and 0.775 on the training set.

## Autograding

```
.AutograderMyTotalScore()
```

```
##
## Step 1:      0/1
## Step 2:      0/1
## Step 3:      0/1
## Step 4:      0/1
## Step 5:      0/1
## Total Score: 0/5
```