

# NEURAL NETWORKS

Jan-Philipp Kolb

03 Juni, 2019

# ARTIFICIAL NEURAL NETWORK FUNDAMENTALS

- ▶ ANNs are engineered computational models inspired by the brain.
- ▶ They are **function approximators**, mapping inputs to outputs,
- ▶ They are composed of many connected computational units, called neurons.
- ▶ Each individual neuron possesses little intrinsic approximation capability;
- ▶ When many neurons function cohesively together, their combined effects show remarkable learning performance.

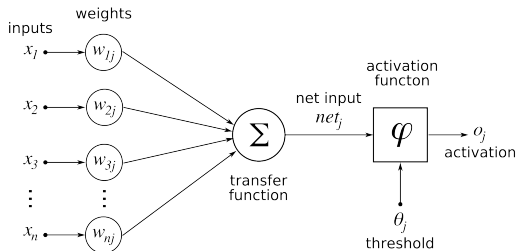
## SOURCE FOR SLIDES

- ▶ Much of the following slides is based on UC Business Analytics R Programming Guide - **neural networks**

# ARTIFICIAL NEURON

- ▶ The transfer function, net input, and activation function correspond to the cell body, and the activation corresponds to the axon and synaptic terminal.

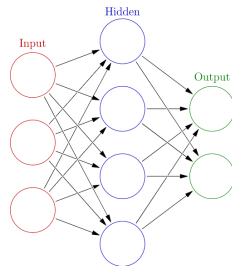
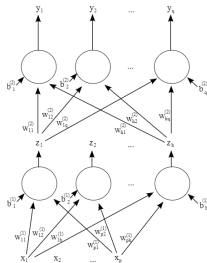
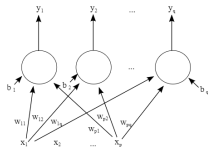
## ARTIFICIAL NEURON—SOURCE: CHRISLB WIKIPEDIA



# THE INPUTS TO THE ARTIFICIAL NEURON

- ▶ ... may correspond to raw data values, or in deeper architectures, may be outputs from preceding artificial neurons.
- ▶ The transfer function sums all the inputs together (cumulative inputs).
- ▶ If the summed input values reach a specified threshold, the activation function generates an output signal (all or nothing).
- ▶ The output signal then moves to a raw output or other neurons depending on specific ANN architecture.
- ▶ This basic artificial neuron is combined with multiple other artificial neurons to create an ANNs such as the ones shown in the following figure.

# EXAMPLES OF MULTI-NEURON



# ANNs DESCRIBED






- ▶ ANNs have an input layer, hidden layer, and output layer.
- ▶ The input layer reads in data values from a user provided input.
- ▶ Within the hidden layer a majority of the “learning” takes place, and the output layer displays the results of the ANN.
- ▶ Each of the red input nodes correspond to an input vector  $x_i$ . Each of the black lines with correspond to a weight,  $w_{ij}^{(l)}$ , and describe how artificial neurons are connections to one another within the ANN.
- ▶ The  $i$  subscript identifies the source and the  $j$  subscript describes to which artificial neuron the weight connects the source to. The green output nodes are the output vectors  $y_q$ .

# ACTIVATION FUNCTIONS

- ▶ The capability of ANNs to learn approximately any function, (given sufficient training data examples) are dependent on the appropriate selection of the **activation function(s)** present in the network.
- ▶ Activation functions enable the ANN to learn non-linear properties present in the data. We represent the activation function here as  $\Phi(\cdot)$ .
- ▶ The input into the activation function is the weighted sum of the input features from the preceding layer.
- ▶ Let  $o_j$  be the output from the  $j$ th neuron in a given layer for a network for  $k$  input vector features.

$$o_j = \Phi(b_j + \sum_{i=1}^p w_i x_i)$$

# ACTIVATION FUNCTIONS

Name	Plot	Equation	Derivative (with respect to $x$ )	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ <sup>[1]</sup>	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
ArSinH		$f(x) = \sinh^{-1}(x) = \ln\left(x + \sqrt{x^2 + 1}\right)$	$f'(x) = \frac{1}{\sqrt{x^2 + 1}}$	$(-\infty, \infty)$



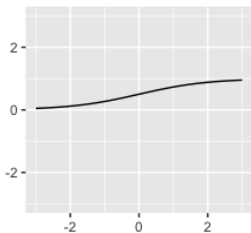
## THE OUTPUT ( $o_j$ )...

- ▶ ... can feed into the output layer of a neural network, or in deeper architectures may feed into additional hidden layers.
- ▶ The activation function determines if the sum of the weighted inputs plus a bias term is sufficiently large to trigger the firing of the neuron.
- ▶ There is not a universal best choice for the activation function, however, researchers have provided ample information regarding what activation functions work well for ANN solutions to many common problems. The choice of the activation function governs the required data scaling necessary for ANN analysis. Below we present activation functions commonly seen in many ANNs.

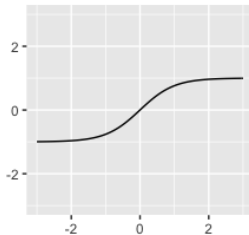
# COMMON ANN ACTIVATION FUNCTIONS

Common ANN Activation Functions

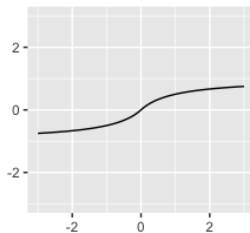
logistic



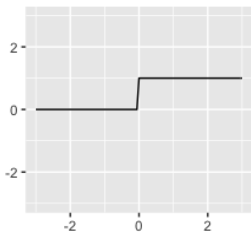
tanh



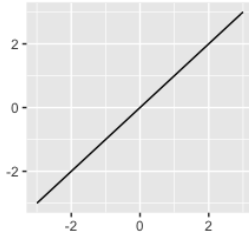
softsign



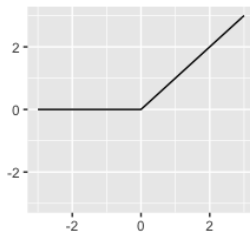
binary step



linear



rectified linear unit (ReLU)



# HOW ANNs LEARN

- ▶ We have described the structure of ANNs, we have not touched on how these networks learn.
- ▶ Assume that we have a data set of labeled observations.
- ▶ We have some features ( $X$ ) describing an output ( $y$ )
- ▶ To begin training our notional single-layer one-neuron neural network we initially randomly assign weights.
- ▶ We then run the neural network with the random weights and record the outputs generated.
- ▶ This is called a forward pass. Output values, in our case called  $y$ , are a function of the input values ( $X$ ), the random initial weights ( $w$ ) and our choice of the threshold function ( $T$ ).

$$\hat{y} = f(X, w, T)$$

# CHOICE OF THE PERFORMANCE FUNCTION

- ▶ Once we have our ANN output values ( $\hat{y}$ ) we can compare them to the data set output values ( $y$ ).
- ▶ To do this we use a performance function  $P$ .
- ▶ The choice of the performance function is a choice of the analyst, we choose to use the One-Half Square Error Cost Function otherwise known as the Sum of Squared Errors (SSE).

# REGRESSION ARTIFICIAL NEURAL NETWORK

- ▶ Regression ANNs predict an output variable as a function of the inputs.
- ▶ Input features (independent variables) can be categorical or numeric types.
- ▶ For regression ANNs, we require a numeric dependent variable.

```
library(tidyverse)
library(neuralnet)
library(GGally)
library(magrittr)
library(readr)
library(dplyr)
```

# DATA PREPARATION

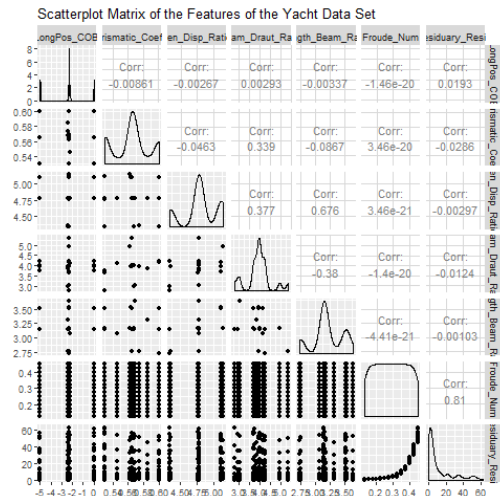
- ▶ Our regression ANN will use the **Yacht Hydrodynamics** data set from UCI's Machine Learning Repository.
- ▶ The yacht data was provided by Dr. Roberto Lopez email.
- ▶ This data set contains data contains results from 308 full-scale experiments performed at the Delft Ship Hydromechanics Laboratory where they test 22 different hull forms.
- ▶ Their experiment tested the effect of variations in the hull geometry and the ship's **Froude number** on the craft's residuary resistance per unit weight of displacement.

# THE DOWNLOAD THE DATA FROM UCI.

```
url<-'http://archive.ics.uci.edu/ml/machine-learning-databases/'
filen <- "00243/yacht_hydrodynamics.data"
colnam <- c('LongPos_COB', 'Prismatic_Coeff', 'Len_Disp_Ratio',
            'Beam_Draut_Ratio', 'Length_Beam_Ratio', 'Froude_Num',
            'Residuary_Resist')
Yacht_Data <- read_table(file = paste0(url,filen),
                        col_names = colnam) %>%
  na.omit()
```

# TAKE A LOOK AT THE DATA SET.

`ggpairs(Yacht_Data, title="Scatterplot Matrix - features of Yacht`





# AN EXCELLENT SUMMARY OF THE VARIATION OF EACH FEATURE

- ▶ Draw your attention to the bottom strip of scatter-plots.
- ▶ This shows the residuary resistance as a function of the other data set features (independent experimental values).
- ▶ The greatest variation appears with the Froude Number feature.
- ▶ It will be interesting to see how this pattern appears in the subsequent regression ANNs.

# DATA SPLIT

Prior to regression ANN construction we first must split the Yacht data set into test and training data sets. - Before we split, first scale each feature to fall in the  $[0, 1]$  interval.

```
# Scale the Data
```

```
scale01 <- function(x){  
  (x - min(x)) / (max(x) - min(x))  
}
```

```
Yacht_Data <- Yacht_Data %>%  
  mutate_all(scale01)
```

```
# Split into test and train sets
```

```
set.seed(12345)
```

```
Yacht_Data_Train <- sample_frac(tbl = Yacht_Data,  
                                replace = FALSE, size = 0.80)
```

```
Yacht_Data_Test <- anti_join(Yacht_Data, Yacht_Data_Train)
```

## THE `SCALE01()` FUNCTION...

- ▶ ... maps each data observation onto the  $[0, 1]$  interval as called in the `dplyr mutate_all()` function.
- ▶ We then provided a seed for reproducible results and randomly extracted (without replacement) 80% of the observations to build the `Yacht_Data_Train` data set.
- ▶ Using `dplyr`'s `anti_join()` function we extracted all the observations not within the `Yacht_Data_Train` data set as our test data set in `Yacht_Data_Test`.

# 1ST REGRESSION ANN

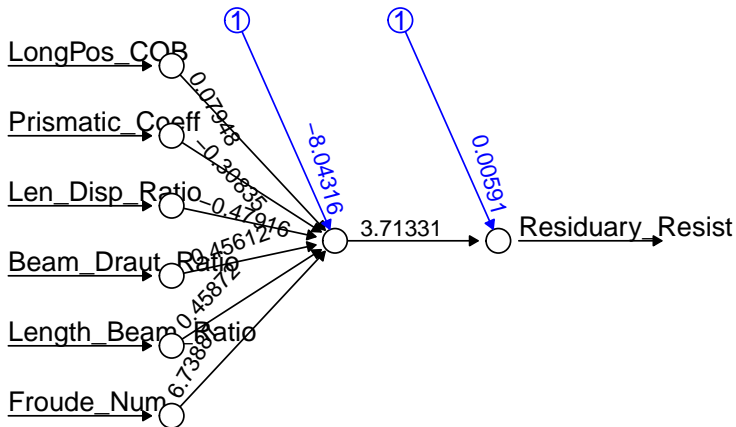
- To begin we construct a 1-hidden layer ANN with 1 neuron, the simplest of all neural networks.

```
set.seed(12321)
Yacht_NN1<-neuralnet(Residuary_Resist~LongPos_COB +
                      Prismatic_Coeff + Len_Disp_Ratio +
                      Beam_Draut_Ratio+
                      Length_Beam_Ratio +Froude_Num,
                      data = Yacht_Data_Train)
```

## THE YACHT\_NN1...

- ... is a list containing all parameters of regression ANN and the results of the neural network on the test data set.

```
plot(Yacht_NN1, rep = 'best')
```



# THE WEIGHTS LEARNED

- ▶ This plot shows the weights learned by the Yacht\_NN1 neural network, and displays the number of iterations before convergence, as well as the SSE of the training data set.
- ▶ To manually compute the SSE you can use the following:

```
NN1_Train_SSE <- sum((Yacht_NN1$net.result-  
                      Yacht_Data_Train[, 7])^2)/2  
paste("SSE: ", round(NN1_Train_SSE, 4))  
## [1] "SSE:  0.0365"
```

# SSE

- ▶ This SSE is the error associated with the training data set.
- ▶ A superior metric for estimating the generalization capability of the ANN would be the SSE of the test data set.
- ▶ The test data set contains observations not used to train Yacht\_NN1 ANN.
- ▶ To calculate the test error, we first run our test observations through the Yacht\_NN1 ANN.
- ▶ This is accomplished with the neuralnet package `compute()` function, which takes as its first input the desired neural network object created by the `neuralnet()` function, and the second argument the test data set feature (independent variable(s)) values.

```
Test_NN1_Output <- compute(Yacht_NN1,  
                           Yacht_Data_Test[, 1:6])$net.result  
(NN1_Test_SSE<-sum((Test_NN1_Output-Yacht_Data_Test[,7])^2)/2)  
## [1] 0.01387174
```

# THE COMPUTE() FUNCTION...

- ▶ ... outputs the response variable, in our case the `Residuary_Resist`, as estimated by the neural network.
- ▶ Once we have the ANN estimated response we can compute the test SSE.
- ▶ Comparing the test error of 0.0084 to the training error of 0.0361 we see that in our case our test error is smaller than our training error.



# REGRESSION HYPERPARAMETERS

- We have constructed the most basic of regression ANNs without modifying any of the default hyperparameters associated with the `neuralnet()` function.

```
# 2-Hidden Layers, Layer-1 4-neurons, Layer-2, 1-neuron,  
# logistic activation function  
set.seed(12321)  
Yacht_NN2 <- neuralnet(Residuary_Resist ~ LongPos_COB +  
  Prismatic_Coeff + Len_Displacement_Ratio +  
  Beam_Draught_Ratio +  
  Length_Beam_Ratio + Froude_Num,  
  data = Yacht_Data_Train,  
  hidden = c(4, 1),  
  act.fct = "logistic")
```

# TRAINING AND TEST ERROR

```
NN2_Train_SSE <- sum((Yacht_NN2$net.result -  
                      Yacht_Data_Train[, 7])^2)/2  
## Test Error  
Test_NN2_Output<-compute(Yacht_NN2,  
                          Yacht_Data_Test[, 1:6])$net.result  
NN2_Test_SSE <- sum((Test_NN2_Output -  
                     Yacht_Data_Test[, 7])^2)/2
```

- ▶ We should try and improve the network by modifying its basic structure and hyperparameter modification.
- ▶ To begin we will add depth to the hidden layer of the network, then we will change the activation function from the logistic to the tangent hyperbolicus (tanh) to determine if these modifications can improve the test data set SSE.

# RESCALE FOR TANH ACTIVATION FUNCTION

- To use the activation function we must rescale the data.

```
scale11 <- function(x) {  
  (2 * ((x - min(x))/(max(x) - min(x)))) - 1  
}  
Yacht_Data_Train <- Yacht_Data_Train %>% mutate_all(scale11)  
Yacht_Data_Test <- Yacht_Data_Test %>% mutate_all(scale11)  
  
# 2-Hidden Layers, Layer-1 4-neurons, Layer-2, 1-neuron,  
# tanh activation function  
set.seed(12321)  
Yacht_NN3 <- neuralnet(Residuary_Resist ~ LongPos_COB +  
  Prismatic_Coeff+Len_Disp_Ratio+Beam_Draut_Ratio +  
    Length_Beam_Ratio + Froude_Num,  
    data = Yacht_Data_Train,  
    hidden = c(4, 1),  
    act.fct = "tanh")
```

# TRAINING AND TEST ERROR

```
## Training Error
```

```
NN3_Train_SSE <- sum((Yacht_NN3$net.result -  
                      Yacht_Data_Train[, 7])^2)/2
```

```
## Test Error
```

```
Test_NN3_Output <- compute(Yacht_NN3,  
                            Yacht_Data_Test[, 1:6])$net.result  
NN3_Test_SSE <- sum((Test_NN3_Output -  
                     Yacht_Data_Test[, 7])^2)/2
```

# 1-HIDDEN LAYER, 1-NEURON, TANH ACTIVATION FUNCTION

```
set.seed(12321)
Yacht_NN4 <- neuralnet(Residuary_Resist ~ LongPos_COB +
                        Prismatic_Coeff + Len_Disp_Ratio +
                        Beam_Draut_Ratio + Length_Beam_Ratio +
                        Froude_Num,
                        data = Yacht_Data_Train,
                        act.fct = "tanh")

## Training Error
NN4_Train_SSE <- sum((Yacht_NN4$net.result -
                      Yacht_Data_Train[, 7])^2)/2
```

# TEST ERROR AND PREPARING A BAR PLOT

```
## Test Error
```

```
Test_NN4_Output <- compute(Yacht_NN4,  
                           Yacht_Data_Test[, 1:6])$net.result  
NN4_Test_SSE<-sum((Test_NN4_Output-Yacht_Data_Test[, 7])^2)/2
```

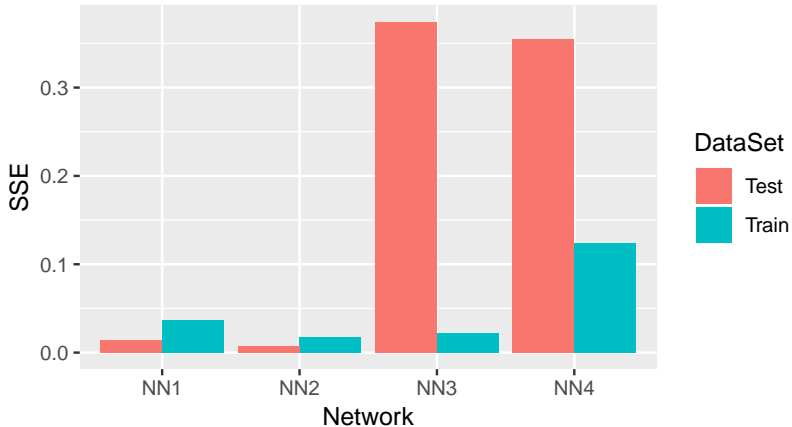
```
# Preparing bar plot of results
```

```
RegNNErr<-tibble(Network=rep(c("NN1", "NN2", "NN3", "NN4"), each=2),  
                  DataSet=rep(c("Train", "Test"), time = 4),  
                  SSE = c(NN1_Train_SSE, NN1_Test_SSE,  
                           NN2_Train_SSE, NN2_Test_SSE,  
                           NN3_Train_SSE, NN3_Test_SSE,  
                           NN4_Train_SSE, NN4_Test_SSE))
```

RegNNErr %>%

```
ggplot(aes(Network, SSE, fill = DataSet)) +  
  geom_col(position = "dodge") +  
  ggtitle("Regression ANN's SSE")
```

Regression ANN's SSE



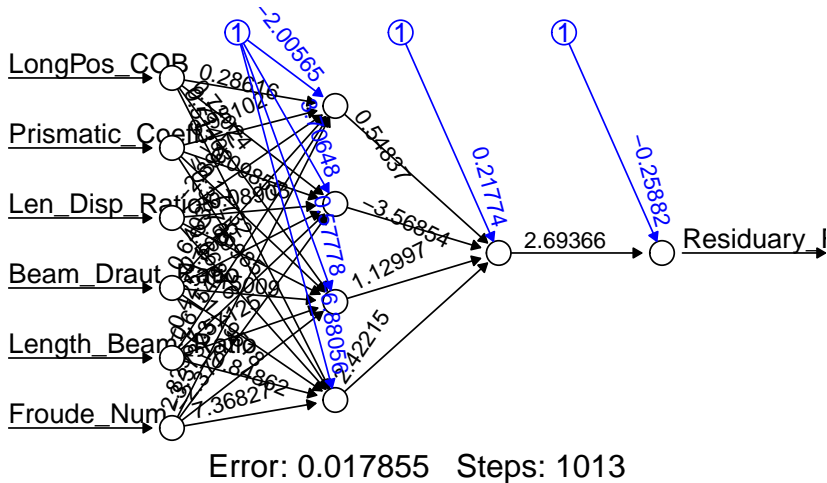
# THE BEST REGRESSION ANN

- ▶ We see that the best regression ANN we found was Yacht\_NN2 with a training and test SSE of 0.0188 and 0.0057.
- ▶ We make this determination by the value of the training and test SSEs only.



## THE STRUCTURE OF YACHT\_NN2

```
plot(Yacht_NN2, rep = "best")
```

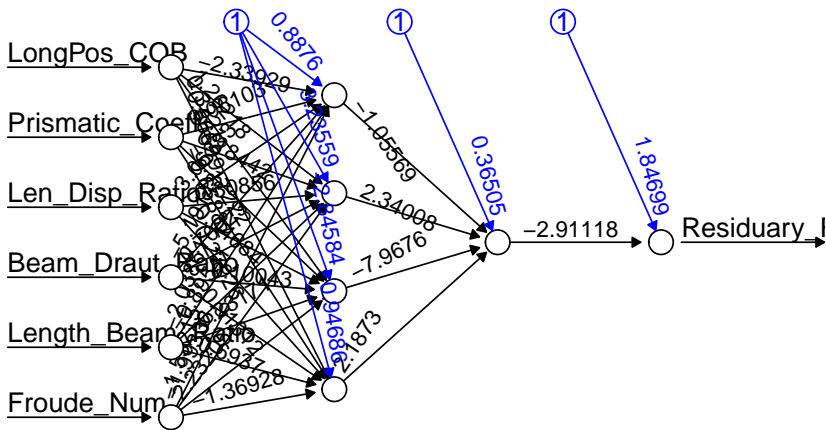


# HYPERPARAMETER SETTINGS

- ▶ We have looked at one ANN for each of the hyperparameter settings.
- ▶ Generally, researchers look at more than one ANN for a given setting of hyperparameters.
- ▶ This capability is built into the `neuralnet` package using the `rep` argument in the `neuralnet()` function.
- ▶ Using the `Yacht_NN2` hyperparameters we construct 10 different ANNs, and select the best of the 10.

```
set.seed(12321)
Yacht_NN2 <- neuralnet(Residuary_Resist ~ LongPos_COB +
                        Prismatic_Coeff + Len_Disp_Ratio +
                        Beam_Draut_Ratio + Length_Beam_Ratio +
                        Froude_Num,data = Yacht_Data_Train,
                        hidden = c(4, 1),
                        act.fct = "logistic",rep = 10)
```

```
plot(Yacht_NN2, rep = "best")
```



Error: 0.032891 Steps: 5909

# DIFFERENT RANDOM SET OF STARTING WEIGHTS

- ▶ By setting the same seed, prior to running the 10 repetitions of ANNs, we force the software to reproduce the exact same Yacht\_NN2 ANN for the first replication.
- ▶ The subsequent 9 generated ANNs, use a different random set of starting weights.
- ▶ Comparing the 'best' of the 10 repetitions, to the Yacht\_NN2, we observe a decrease in training set error indicating we have a superior set of weights.

# WRAPPING UP - REGRESSION ARTIFICIAL NEURAL NETWORK

- ▶ The `neuralnet` package we used so far is one of many tools available for ANN implementation in R.

## OTHERS INCLUDE:

- ▶ `nnet`
- ▶ `autoencoder`
- ▶ `caret`
- ▶ `RSNNS`
- ▶ `h2o`

# CLASSIFICATION ARTIFICIAL NEURAL NETWORK

- ▶ Classification ANNs seek to classify an observation as belonging to some discrete class as a function of the inputs.
- ▶ The input features (independent variables) can be categorical or numeric types.
- ▶ We need a categorical feature as the dependent variable.

# PREPARATION - CLASSIFICATION ARTIFICIAL NEURAL NETWORK

- ▶ The following slides are based on UC Business Analytics R Programming Guide on **Classification Artificial Neural Network**

```
library(tidyverse)
library(neuralnet)
library(GGally)
```

# DATA PREPARATION

- ▶ Our classification ANN will use **Haberman's Survival** data set from UCI's Machine Learning Repository.
- ▶ Haberman's data set was provided by Tjen-Sien Lim, and contains cases from a 1958 and 1970 study conducted at the University of Chicago's Billings Hospital on the survival of 306 patients who had undergone surgery for breast cancer.
- ▶ We will use this data set to predict a patient's 5-year survival as a function of their age at date of operation, year of the operation, and the number of positive axillary nodes detected.



# DOWNLAOD OF THE DATA

- ▶ We first download the data from UCI. When this data is imported, the `Survival` feature is imported as an integer, this needs to be a categorical logical value so we will modify this feature using the `mutate()` function in the `dplyr` package.

## MEANING OF SURVIVAL FEATURE

- ▶ A value of 1 - patient survived for at least 5 years after the operation
- ▶ A value of 0 - the patient died within 5 years.

```
url1<-'http://archive.ics.uci.edu/ml/machine-learning-'  
url2<-"databases//haberman/haberman.data"  
url <- paste0(url1,url2)  
Hab_Data<-read_csv(file=url,col_names=c('Age', 'Operation_Year',  
                                         'Number_Pos_Nodes','Survival'))
```

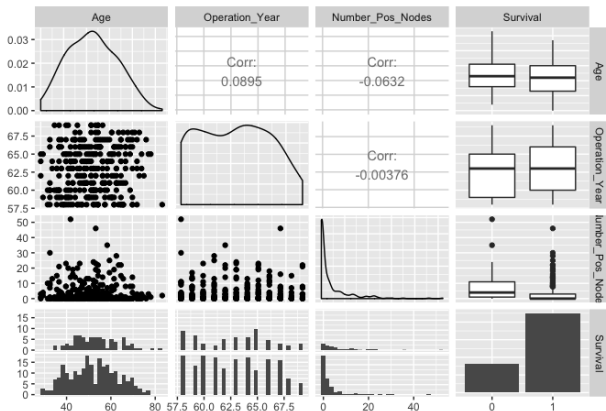
# MUTATE THE SURVIVAL VARIABLE

```
Hab_Data <- Hab_Data%>%  
  na.omit() %>%  
  mutate(Survival = ifelse(Survival == 2, 0, 1),  
         Survival = factor(Survival))
```

# SCATTERPLOT MATRIX

```
ggpairs(Hab_Data, title="Scatterplot Matrix Haberman's  
Survival Features")
```

Scatterplot Matrix of the Features of the Haberman's Survival Data Set



# INTERPRETING THE SCATTERPLOT

- ▶ Many patients survived at least 5 years after the operation.
- ▶ Of the patients that survived (bottom-subplots of the Survival row in the Scatterplot Matrix), we see many of the patients have few numbers of positive axillary nodes detected.
- ▶ Examination of the Age feature shows a few of the most elderly patients died within 5 years, and of the youngest patients we see increased 5-year survivability.
- ▶ We forego any more detailed visual inspection in favor of learning the relationships between the features using our classification ANN.

# AIC AND BIC FOR FINAL MODEL SELECTION

- ▶ We must scale our features to fall on a closed  $[0, 1]$  interval.
- ▶ For classification ANNs using `neuralnet`, we will not use a training and test set for model evaluation, instead we will use the AIC and BIC for final model selection.
- ▶ These metrics balance the error as a function of the total number of model parameters, in the ANN case the parameters correspond to the total number of hidden nodes.

```
scale01 <- function(x){  
  (x - min(x)) / (max(x) - min(x))  
}
```

```
Hab_Data <- Hab_Data %>%  
  mutate(Age = scale01(Age),  
         Operation_Year = scale01(Operation_Year),  
         Number_Pos_Nodes = scale01(Number_Pos_Nodes),  
         Survival = as.numeric(Survival)-1)
```

# CLASSIFICATION ANNs IN THE NEURALNET PACKAGE...

- ... require that the response feature, in this case Survival, be inputted as a Boolean feature. We modify the feature then run the initial classification ANN.

```
Hab_Data <- Hab_Data %>%  
  mutate(Survival = as.integer(Survival) - 1,  
         Survival = ifelse(Survival == 1, TRUE, FALSE))
```

# 1ST CLASSIFICATION ANN

- ▶ We construct a 1-hidden layer ANN with 1 neuron.
- ▶ The `neuralnet` package defaults to random initial weight values, for reproducibility we set a seed and construct the network.
- ▶ We have added three additional arguments for the classification ANN using the `neuralnet` package, `linear.output`, `err.fct`, and `likelihood`.
- ▶ Setting the `linear.output` to `FALSE` and the `err.fct` to `"ce"` indicates that we are performing a classification, and forces the model to output what we may interpret as a probability of each observation belonging to Survival class 1.

# CLASSIFICATION ANN WITH CROSS-ENTROPY ERROR METRIC

- ▶ For classification ANN the cross-entropy error metric is more appropriate than the SSE used in regression ANNs.
- ▶ The likelihood argument set to TRUE indicates to neuralnet that we would like to see the AIC and BIC metrics.

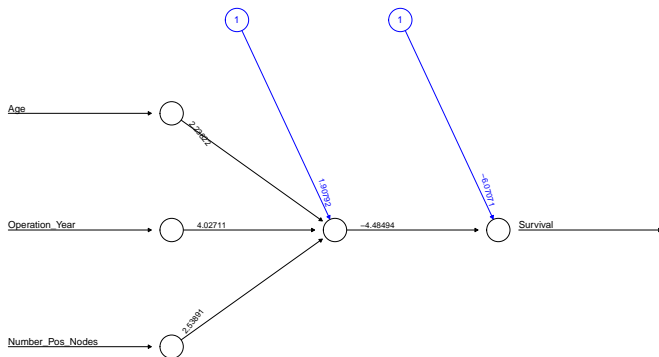
```
set.seed(123)
Hab_NN1 <- neuralnet(Survival ~ Age + Operation_Year +
                     Number_Pos_Nodes,
                     data = Hab_Data,
                     linear.output = FALSE,
                     err.fct = 'ce',
                     likelihood = TRUE)
```



## DIAGRAM OF THE HAB\_NN1

The Hab\_NN1 is a list containing all parameters of the classification ANN as well as the results of the neural network on the test data set. - To view a diagram of the Hab\_NN1 use the `plot()` function.

```
plot(Hab_NN1, rep = 'best')
```



Error: 0.008623 Steps: 63

# THE CROSS-ENTROPY ERROR

- ▶ The error displayed in this plot is the cross-entropy error, which is a measure of the differences between the predicted and observed output for each of the observations in the Hab\_Data data set.
- ▶ To view the Hab\_NN1 AIC, BIC, and error metrics run the following.

```
Hab_NN1_Train_Error <- Hab_NN1$result.matrix[1,1]  
paste("CE Error: ", round(Hab_NN1_Train_Error, 3))
```

```
## [1] "CE Error:  0.009"
```

```
Hab_NN1_AIC <- Hab_NN1$result.matrix[4,1]  
paste("AIC: ", round(Hab_NN1_AIC,3))
```

```
## [1] "AIC:  12.017"
```

```
Hab_NN2_BIC <- Hab_NN1$result.matrix[5,1]  
paste("BIC: ", round(Hab_NN2_BIC, 3))
```

```
## [1] "BIC:  34.359"
```

# CLASSIFICATION HYPERPARAMETERS

- ▶ Classification ANNs within the `neuralnet` package require the use of the `ce` error.
- ▶ This forces us into using the default `act.fun` hyperparameter value.
- ▶ As a result we will only change the structure of the classification ANNs using the hidden function setting.

```
set.seed(123)
# 2-Hidden Layers, Layer-1 2-neurons, Layer-2, 1-neuron
Hab_NN2 <- neuralnet(Survival ~ Age + Operation_Year +
                     Number_Pos_Nodes,data = Hab_Data,
                     linear.output = FALSE,err.fct = 'ce',
                     likelihood = TRUE, hidden = c(2,1))
```

## 2-HIDDEN LAYERS, LAYER-1 2-NEURONS, LAYER-2, 2-NEURONS

```
set.seed(123)
Hab_NN3<-neuralnet(Survival~Age+Operation_Year+
                    Number_Pos_Nodes,
                    data = Hab_Data,linear.output = FALSE,err.fct = 'ce',
                    likelihood = TRUE, hidden = c(2,2))
```

## 2-HIDDEN LAYERS, LAYER-1 1-NEURON, LAYER-2, 2-NEURON

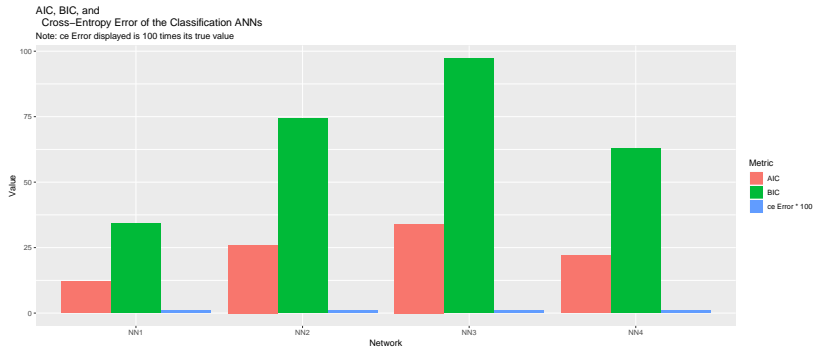
```
set.seed(123)
Hab_NN4 <- neuralnet(Survival ~ Age + Operation_Year +
  Number_Pos_Nodes,data = Hab_Data,
  linear.output = FALSE,err.fct = 'ce',
  likelihood = TRUE,hidden = c(1,2))
```

## BAR PLOT OF RESULTS

```
Class_NN_ICs<-tibble('Network'=rep(c("NN1","NN2","NN3","NN4"),
  each = 3), 'Metric'=rep(c('AIC', 'BIC', 'ce Error * 100'),
  length.out = 12), 'Value' = c(Hab_NN1$result.matrix[4,1],
  Hab_NN1$result.matrix[5,1], 100*Hab_NN1$result.matrix[1,1],
  Hab_NN2$result.matrix[4,1], Hab_NN2$result.matrix[5,1],
  100*Hab_NN2$result.matrix[1,1], Hab_NN3$result.matrix[4,1],
  Hab_NN3$result.matrix[5,1], 100*Hab_NN3$result.matrix[1,1],
  Hab_NN4$result.matrix[4,1], Hab_NN4$result.matrix[5,1],
  100*Hab_NN4$result.matrix[1,1]))
```

# AIC, BIC, AND CROSS-ENTROPY ERROR

```
Class_NN_ICs %>%ggplot(aes(Network,Value,fill=Metric)) +  
  geom_col(position = 'dodge')+ggtitle("AIC, BIC, and  
  Cross-Entropy Error of the Classification ANNs",  
  "Note: ce Error displayed is 100 times its true value")
```



# OCCAM'S RAZOR

- ▶ The plot indicates that as we add hidden layers and nodes within those layers, our AIC and cross-entropy error grows.
- ▶ The BIC appears to remain relatively constant across the designs.
- ▶ Here we have a case where Occam's razor clearly applies, the 'best' classification ANN is the simplest.



# EXERCISE NEURAL NETWORKS (I)

## CREATE EXAMPLE DATA

- 0) The R-package `nnet` has the capacity to build classification ANNs. Install and take a look at the documentation of `nnet` to see how it compares with `neuralnet`.
- 1) Set the seed to 42, draw 200 numbers from a unified random distribution between -10 and 10 and store them in a vector named `x`.
- 2) Create a vector named `y` containing the value of `sin(x)`.

## RANDOMIZE INITIAL WEIGHTS

- 3) Create a vector of 10 random values, picked in the interval `[-1,1]`.

# EXERCISE NEURAL NETWORKS (II)

## SPLIT THE DATASET

- 4) Neural networks have a strong tendency of overfitting data, they become really good at describing the relationship between the values in the data set, but are not effective with data that wasn't used to train your model. As a consequence, we need to cross-validate our model. Set the seed to 42, then create a training set containing 75% of the values in your initial data set and a test set containing the rest of your data.

## CREATE A FIRST MODEL

- 5) Load the `nnet` package and use the function `nnet` to create a model. Pass your weights via the `Wts` argument and set `maxit=50`. Set `linout=T` and take some time to look at the structure of your model.

# EXERCISE NEURAL NETWORKS (III)

## PREDICTION

- 6) Predict the output for the test set and compute the RMSE of your predictions. Plot the function  $\sin(x)$  and then plot your predictions.

## REPEAT WITH DIFFERENT PARAMETERS

- 7) The number of neurons in the hidden layer, and the number of hidden layer used, has a great influence on the effectiveness of your model. Repeat the exercises four to six, but this time use a hidden layer with seven neurons and initiate randomly 22 weights.

# EXERCISE NEURAL NETWORKS (IV)

## NORMALIZE

- 8) Now let us use neural networks to solve a classification problem, so let's load the iris data set! It is good practice to normalize your input data to uniformize the behavior of your model over different range of value and have a faster training. Normalize each factor so that they have a mean of zero and a standard deviation of 1, then create your train and test set.

## CREATE MODEL WITH NNET

- 9) Use the `nnnet()` and use a hidden layer of ten neurons to create your model. We want to fit a function which have a finite amount of value as output. To do so, set the `linout` argument to true. Look at the structure of your model. With classification problem, the output is usually a factor that is coded as multiple dummy variables, instead of a single numeric value. As a consequence, the output layer have as one less neuron than the number of levels of the output factor.

# EXERCISE NEURAL NETWORKS (V)

## PREDICTION AND CONFUSION TABLE

- 10) Make prediction with the values of the test set.
- 11) Create the confusion table of your prediction and compute the accuracy of the model.

# WHAT ARE THE ADVANTAGES AND DISADVANTAGES OF NEURAL NETWORKS?

## ADVANTAGES:

- ▶ Neural networks (specifically deep NNs) have led to performance breakthroughs for unstructured datasets such as images, audio, and video. Their incredible flexibility allows them to learn patterns that no other ML algorithm can learn.

## DISADVANTAGES:

- ▶ Neural networks require a large amount of training data to converge. It's also difficult to pick the right architecture, and the internal "hidden" layers are incomprehensible.

# ADDITIONAL RESOURCES AND FURTHER READING

- ▶ Neural networks with R
- ▶ How to handle imbalanced classes
- ▶ Free data science resources