Neural Networks

Jan-Philipp Kolb

20 Mai, 2019

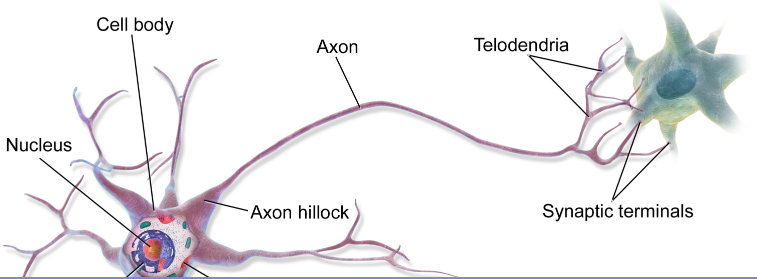
Artificial Neural Network Fundamentals

- Artificial neural networks (ANNs) describe a specific class of machine learning algorithms designed to acquire their own knowledge by extracting useful patterns from data.
- ANNs are function approximators, mapping inputs to outputs, and are composed of many interconnected 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.

Biologic Model

- ANNs are engineered computational models inspired by the brain (human and animal).
- While some researchers used ANNs to study animal brains, most researchers view neural networks as being inspired by, not models of, neurological systems.
- The following shows the basic functional unit of the brain, a biologic neuron.

Biologic Neuron–Source: Bruce Blaus **Wikipedia**



ANN neurons

- ANN neurons are simple representations of their biologic counterparts.
- In the biologic neuron figure please note the Dendrite, Cell body, and the Axon with the Synaptic terminals.
- In biologic systems, information (in the form of neuroelectric signals) flow into the neuron through the dendrites.
- If a sufficient number of input signals enter the neuron through the dendrites, the cell body generates a response signal and transmits it down the axon to the synaptic terminals.
- The specific number of input signals required for a response signal is dependent on the individual neuron.
- When the generated signal reaches the synaptic terminals neurotransmitters flow out of the synaptic terminals and interact with dendrites of adjoining neurons.

Three major takeaways from the biologic neuron

1.) The neuron only generates a signal if a sufficient number of input signals enter the neurons dendrites (all or nothing)

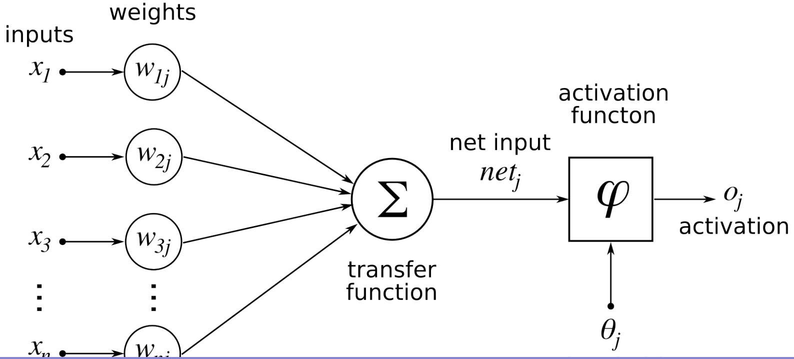
2.) Neurons receive inputs from many adjacent neurons upstream, and can transmit signals to many adjacent signals downstream (cumulative inputs)

3.) Each neuron has its own threshold for activation (synaptic weight).

Artifical Neuron

- The artificial analog of the biologic neuron is shown below.
- In the artificial model the inputs correspond to the dendrites, the transfer function, net input, and activation function correspond to the cell body, and the activation corresponds to the axon and synaptic terminal.

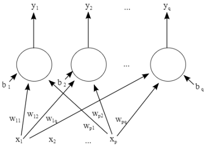
Artifical Neuron–Source: Chrislb Wikipedia

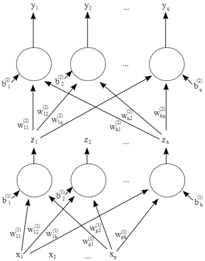


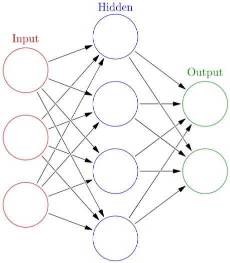
The inputs to the artificial neuron

- ... 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–Source: Click images







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 is where a majority of the "learning" takes place, and the output layer displays the results of the ANN.
- In the bottom plot of the figure, 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.

Examination of the figure

- Examination of the figure's top-left and top-right plots show two possible ANN configurations.
- In the top-left, we see a network with one hidden layer with q artificial neurons, p input vectors x, and generates q output vectors y.
- The bias term is a simple constant valued 1 to each hidden node acting akin to the grand mean in a simple linear regression.
- Each bias term in a ANN has its own associated weight w.
- In the top-right ANN we have a network with two hidden layers.
- This network adds superscript notation to the bias terms and the weights to identify to which layer each term belongs.
- Weights and biases with a superscript 1 act on connecting the input layer to the first layer of artificial neurons and terms with a superscript 2 connect the output of the second hidden layer to the output vectors.
- The size and structure of ANNs are only limited by the imagination of the analyst.

Activation Functions

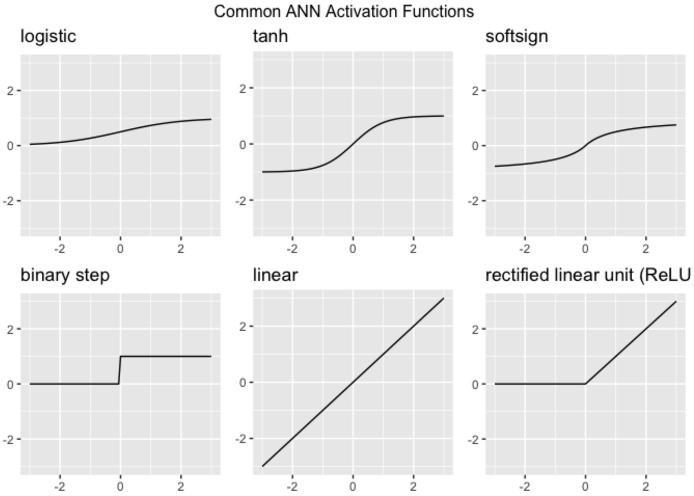
- The capability of ANNs to learn approximately any function, (given sufficient training data examples) are dependent on the appropriate selection of the Activation Function(s) present in the network.
- Activation functions enable the ANN to learn non-linear properties present in the data. We represent 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 jth neuron in a given layer for a network for k input vector features.

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

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 may ANNs.

Common ANN Activation functions



How ANNs Learn

We have described the structure of ANNs, however, we have not touched on how these networks learn. - For the purposes of this discussion we assume that we have a data set of labeled observations. - Data sets in which we have some features (X) describing an output (y) fall under machine learning techniques called Supervised Learning. - 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)
```

Data Preparation

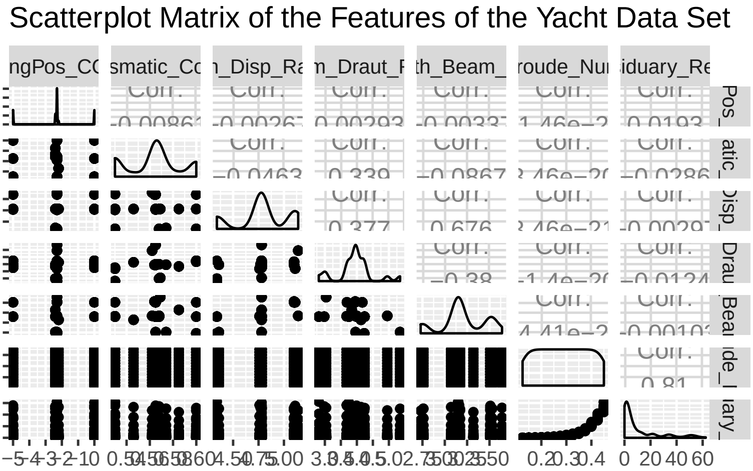
- Our regression ANN will use the **Yacht Hydrodynamics** data set from UCI's Machine Learning Repository.
- The yacht data was provided by Dr. Roberto Lopez email.
- This data set contains data contains results from 308 full-scale experiments performed at the Delft Ship Hydromechanics Laboratory where 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

Yacht_Data <- read_table(file = url,
                         col_names = c('LongPos_COB', 'Prismatic
                                       'Len_Disp_Ratio', 'Beam_D
                                       'Length_Beam_Ratio','Frou
                                       'Residuary_Resist')) %>%
  na.omit()
```

Take a look at the data set.

```
ggpairs(Yacht_Data, title = "Scatterplot Matrix of the Features
```



An excellent summary of the variation of each feature

- Draw your attention to the bottom-most 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 = FALS
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

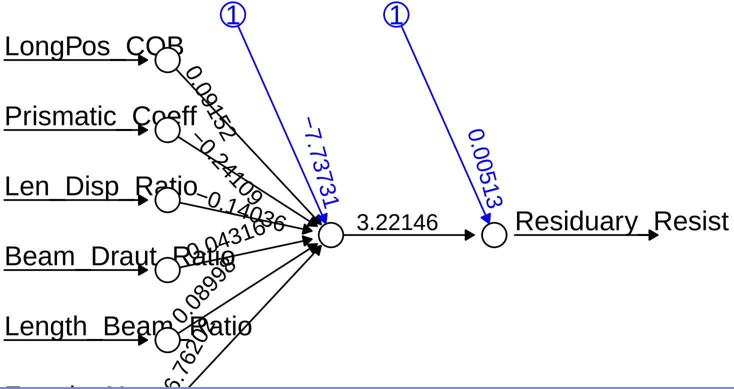
- To begin we construct a 1-hidden layer ANN with 1 neuron, the simplest of all neural networks.

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

The Yacht_NN1...

- ... is a list containing all parameters of the regression ANN as well as the results of the neural network on the test data set.
- To view a diagram of the Yacht_NN1 use the plot() function.

```
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[,
paste("SSE: ", round(NN1_Train_SSE, 4))
```

```
## [1] "SSE:  0.0361"
```

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.
- Recall, the test data set contains observations not used to train the Yacht_NN1 ANN.
- To calculate the test error, we first must 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])$ne
NN1_Test_SSE <- sum((Test_NN1_Output - Yacht_Data_Test[, 7])^2)/
NN1_Test_SSE
```

```
## [1] 0.008417631
```

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

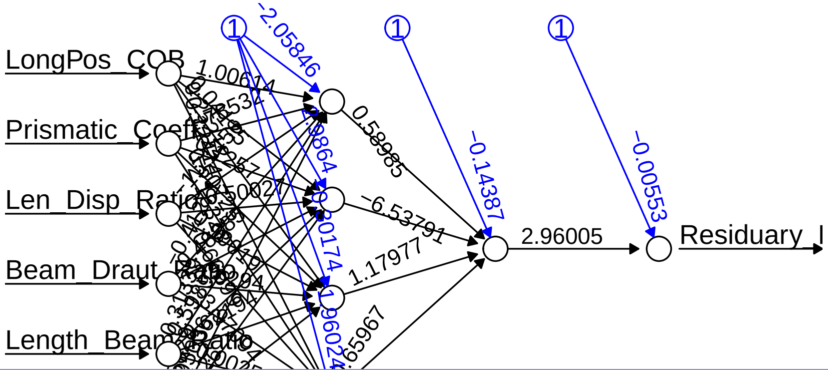
- We have constructed the most basic of regression ANNs without modifying any of the default hyperparameters associated with the neuralnet() function.

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

## Training Error
NN2_Train_SSE <- sum((Yacht_NN2$net.result - Yacht_Data_Train[,

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

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. Yacht_NN2's structure is presented here:

```
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 + Prismati
                       data = Yacht_Data_Train,
                       hidden = c(4, 1),
                       act.fct = "logistic",
                       rep = 10)

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

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

- In the next tutorial we will cover classification ANNs.
- The `neuralnet` package used in this tutorial is one of many tools available for ANN implementation in R.

Others include:

- nnet
- autoencoder
- caret
- RSNNS
- h2o

Exercise

- Why do we split the yacht data into a training and test data sets?
- Re-load the Yacht Data from the UCI Machine learning repository yacht data without scaling. Run any regression ANN. What happens? Why do you think this happens?
- After completing exercise question 1, re-scale the yacht data. Perform a simple linear regression fitting Residuary_Resist as a function of all other features. Now run a regression neural network (see 1st Regression ANN section). Plot the regression ANN and compare the weights on the features in the ANN to the p-values for the regressors.
- Build your own regression ANN using the scaled yacht data modifying one hyperparameter. Use ?neuralnet to see the function options. Plot your ANN.

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.

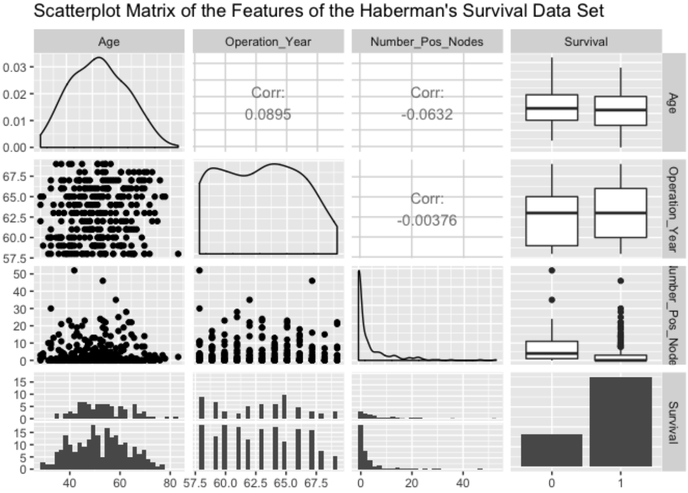
Meaning of Survival feature

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

```
url <- 'http://archive.ics.uci.edu/ml/machine-learning-databases

Hab_Data <- read_csv(file = url,
                     col_names = c('Age', 'Operation_Year',
                                   'Number_Pos_Nodes','Survival'
  na.omit() %>%
  mutate(Survival = ifelse(Survival == 2, 0, 1),
         Survival = factor(Survival))
```

Scatterplot Matrix

```
ggpairs(Hab_Data, title = "Scatterplot Matrix of the Features of
```



shows that many more 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

- As in the regression ANN, we must scale our features to fall on the closed $[0, 1]$ interval.
- For classification ANNs using the `neuralnet` package 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 model 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)
```

1st Classification ANN

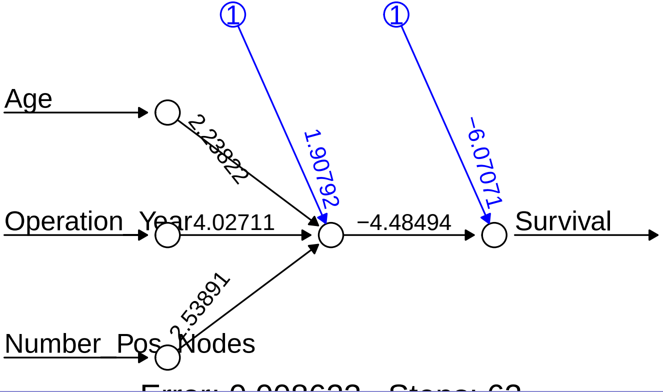
- We construct a 1-hidden layer ANN with 1 neuron.
- The `neuralnet` package defaults to random initial weight values, for reproducibility we set a seed and construct the network.
- We have added three additional arguments for the classification ANN using the `neuralnet` package, `linear.output`, `err.fct`, 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.
- 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.

```r
set.seed(123)
Hab_NN1 <- neuralnet(Survival ~ Age + Operation_Year + Number_Po
                     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')
```



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_Po
                     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 <- Hab_NN2 <- neuralnet(Survival ~ Age + Operation_Year
```

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.

Wrapping Up

- Keep in mind that the `neuralnet` package used in this tutorial is only one of many tools available for ANN implementation in R. Others include:
- `nnet`
- `autoencoder`
- `caret`
- `RSNNS`
- `h2o`

Exercise

1.) Build your own classification ANN using the Hab_Data data set.

2.) The iris data set contains 4 numeric features describing 3 plant species. Think about how we would need to modify the iris data set to prepare it for a classification ANN. Hint, the data set for classification will have 7 total features.

3.)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.

Additional resources and further reading

- Neural networks with R

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: However, they 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.

```
set.seed(500)
library(MASS)
data <- Boston
```

```
index <- sample(1:nrow(data),round(0.75*nrow(data)))
train <- data[index,]
test <- data[-index,]
lm.fit <- glm(medv~., data=train)
summary(lm.fit)
```

```
##
## Call:
## glm(formula = medv ~ ., data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -14.9143   -2.8607   -0.5244    1.5242   25.0004
##
```

Preparing to fit the neural network

- It is good practice to normalize your data before training a neural network.

```
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)
scaled <- as.data.frame(scale(data, center = mins, scale = maxs
train_ <- scaled[index,]
test_ <- scaled[-index,]
```

Parameters

```
install.packages("neuralnet")

library(neuralnet)
n <- names(train_)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collaps
nn <- neuralnet(f,data=train_,hidden=c(5,3),linear.output=T)
```

A plot

```
plot(nn)
```

Further Links

- How to handle inbalanced classes
- Free data science resources