Assignment 2

Data Analytics Foundation
Mandeep Sarangal
251000108

Description of the forecasting problem and data

Weather data for the city of Szeged in Hungary, from 2006 to 2016, is available. This forecasting problem deals with predicting the **Apparent Temperature** value given the values of 4 attributes, which are **Humidity, Temperature, Wind speed and Pressure** on that day.
Y = Apparent Temperature
X = Humidity, Temperature, Wind speed and Pressure

Where X represents all the independent variables/attributes, also known as predictor variables. Based on the values of X, we will predict the values of Y which is our variable of interest.

Description of available data:

Column name	**Type**	**Description**
Formatted Date	DateTime	Date and Time of the day
Summary	String	Short Summary of the day
Precip type	String	Type of precipitation
Temperature	Numeric	Actual Temperature
Apparent Temperature	Numeric	Temperature perceived by humans
Humidity	Numeric	Value of humidity
Wind Speed	Numeric	Speed of Wind
Wind Bearing	Numeric	Direction of the Wind
Visibility	Numeric	Distance at which an object or light can be clearly discerned
Loud Cover	Numeric	Total cover
Pressure	Numeric	Value of atmospheric pressure

Daily Summary	String	Overall summary for the day

Link to Dataset : https://www.kaggle.com/budincsevity/szeged-weather/data

Overview of network architecture and parameters that were tuned

Feed Forward Neural Network (FFNN) is the selected architecture for this problem.

FFNN form the basis of many important Neural Networks being used in the recent times, such as Convolutional Neural Networks (used extensively in computer vision applications), Recurrent Neural Networks (widely used in Natural language understanding and sequence learning) and so on.

FFNN can be used for classification as well as regression problems. In my case, I am using it for regression to predict values of a continuous variable. Several parameters like number of hidden layers, learning rate, activation function etc. can be tuned for optimization.

Tuned parameters for this specific problem:

- *Hidden Layers*
  A thumb rule for deciding the number of hidden layers required in a problem statement is as follows:
  $\sqrt{Number\ of\ attributes\ in\ input\ layer} = \sqrt{4} = 2$

  So I experimented with 1 and 2 hidden layers at a given time.

  Also, if the data is linear, it does not need any hidden layers at all. But if the nature of data is more complex It will need hidden layers. Each progressive hidden layer captures increasingly more complex features and patterns in the data which help making accurate predictions in the output layer.

- *Neurons*
  Deciding the number of neurons in the hidden layers is a very important part of deciding your overall neural network architecture. Using too few neurons in the hidden layers will result in something called Underfitting. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set.

  Using too many neurons in the hidden layer can cause two problems. First, it can result in Overfitting the data. Overfitting is a condition when the neural network has so much information processing capacity that training set is too small to train all the neurons in the hidden layers. Second, even if the training data is sufficient, large number of neurons might drastically increase the training time of the neural network, which is not desirable.

Ultimately, the selection of number of neurons in each hidden layer neural network comes down to trial and error.

- *Back propagation Algorithm*
  I tried 2 variations of back propagation algorithms, namely:
  1. Normal Back propagation (algorithm='backprop')
  2. Resilient Back propagation (algorithm=' rprop+')

  Resilient Back propagation gave a better result by reducing the error value

- *Activation Function*
  Choosing a good activation function allows training better and efficiently.

  I tried 3 variations of activation functions
  1. Sigmoid: It is a probabilistic function and is most useful in classification problems
  2. TanH: This activation function gave me the best result. The resulting error was the lowest in this case.
  3. Relu: It is the most commonly used function but in this particular problem, the results given were not optimal.

Description of the process, including, data pre-processing, model training and evaluation.

Step 1

In order to select relevant independent attributes, I calculated the Correlation Co-efficient for all the independent variables among each other. Only those attributes were selected for whom the correlation co-efficient was not strong on either sides i.e. it was neither very close to 1 nor to -1. It's a good practice to choose independent attributes which are not strongly linear in relationship to each other.

If any 2 independent attributes had a very high correlation co-efficient, one of them was dropped.

After above analysis, 4 attributes were shortlisted for further data processing:

- Temperature
- Humidity
- Wind Speed
- Pressure

Step 2

Data Normalization

For the optimization to proceed numerically smooth, all the attribute values were normalized using min max scaling. Normalizing essentially means scaling the values such that they have zero mean and equal variance.

Step 3

The dataset was divided into Training, Validation and Test datasets in the ratio 3:1:1 respectively.

Training data was used to train the Algorithm
Validation data was used to validate the predictions of the trained neural network model
Test data was used for predictions after the model was trained (Training data) and optimized (Validation data) to minimize the error rate.

Step 4

Evaluation Criteria: Mean Squared Error (MSE)

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

The mean squared error tells us how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them. The squaring is necessary to remove any negative signs. It also gives more weight to larger differences. It’s called the **mean** squared error as we are finding the average of a set of errors.
Every time the model was trained by tuning different parameters, MSE was calculated using the predicted and actual values. The main focus was on minimizing the MSE throughout the process of model training.

Results, process of reaching final accuracy, demonstrated with graphs and observations

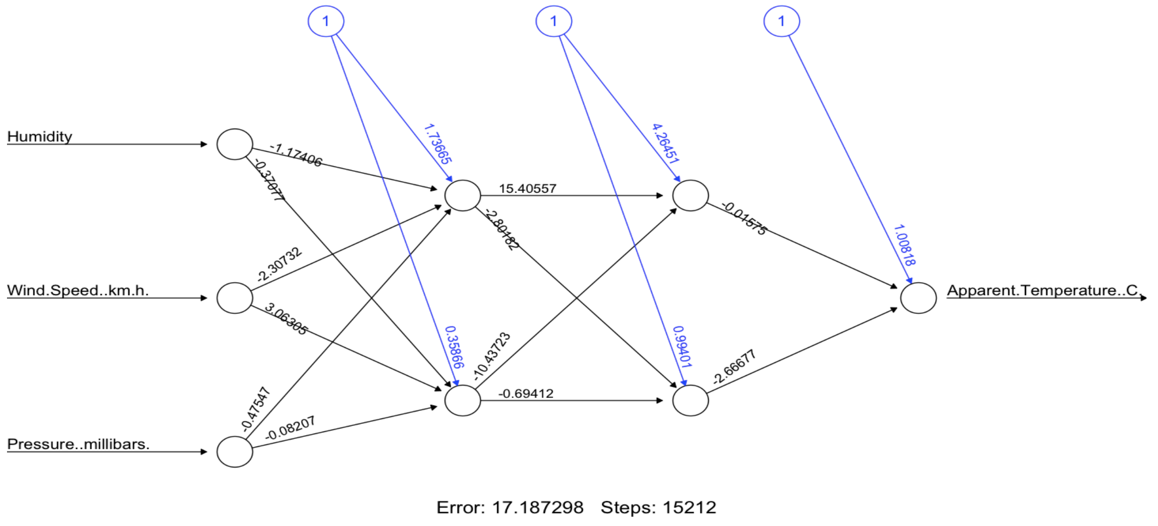
First Observation

Independent Variables	**Correlation with Apparent Temperature**
Temperature	**0.99301565**
Humidity	**-0.64216355**
Wind Speed	**0.05747613**
Pressure	**-0.15897986**

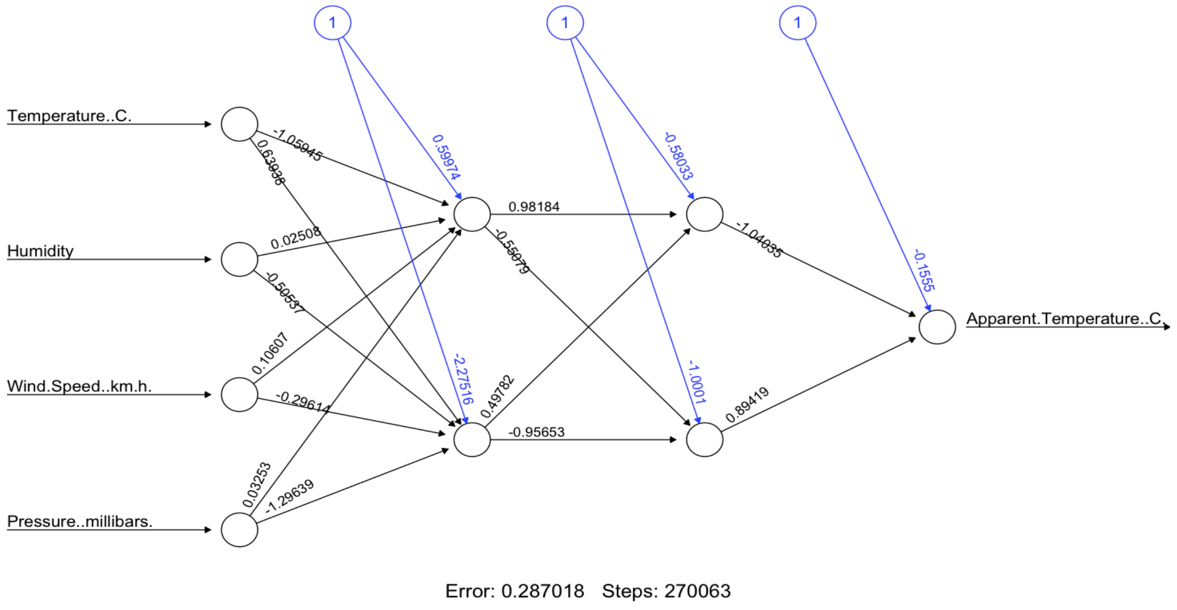
The above table clearly shows that the correlation between Temperature and Apparent Temperature is really high. This means that while training the neural network model, the influence of Temperature variable will be the greatest.

So just to see this in action, I trained the NN model with and without Temperature attribute.

Without Temperature



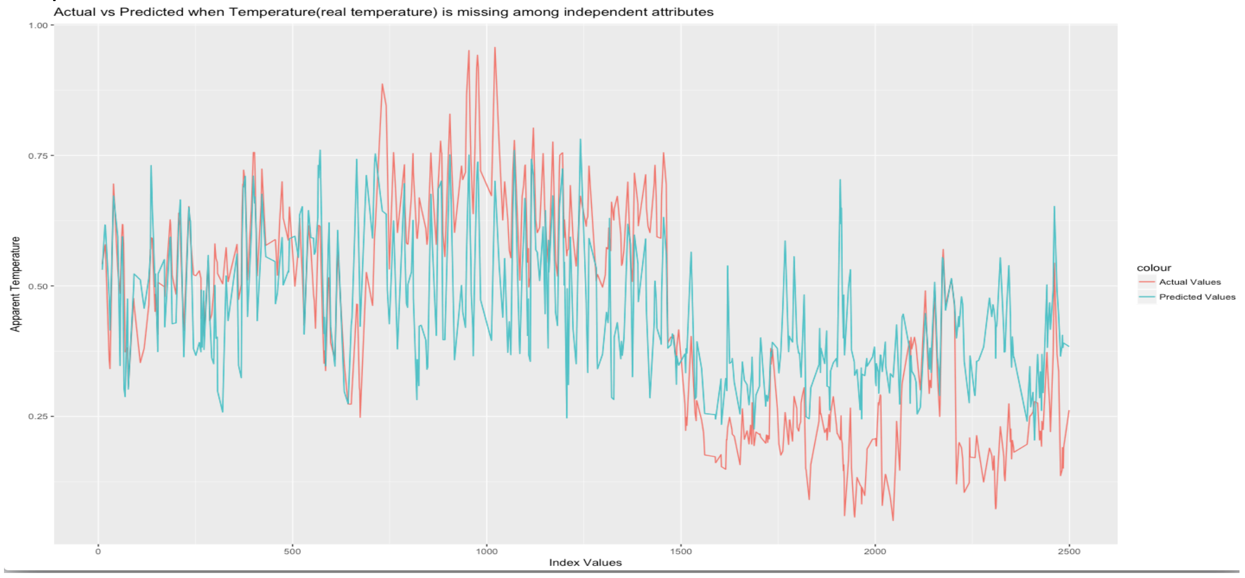
With Temperature



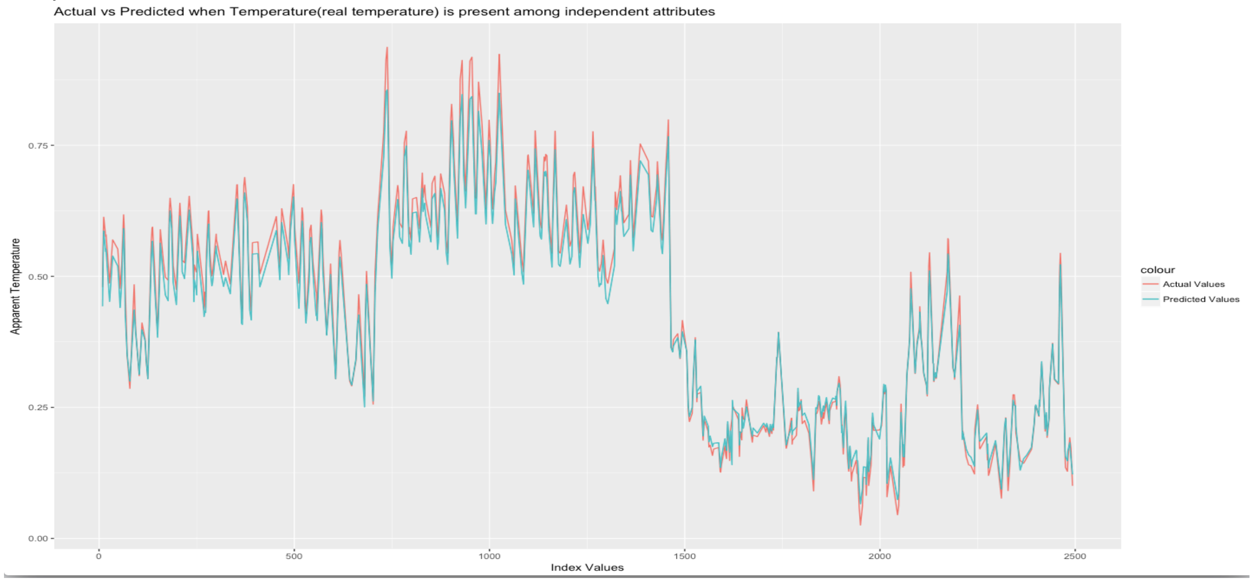
It can be clearly seen that after adding Temperature as one of the independent variables, <u>the error drops from 17.187298 to 0.287018</u>. This proves that Temperature attribute alone played a vital role in minimizing the cost function to its minimum value and as a result gave a very small value of error.

- ● Actual Values
- ● Predicted values

Graph A



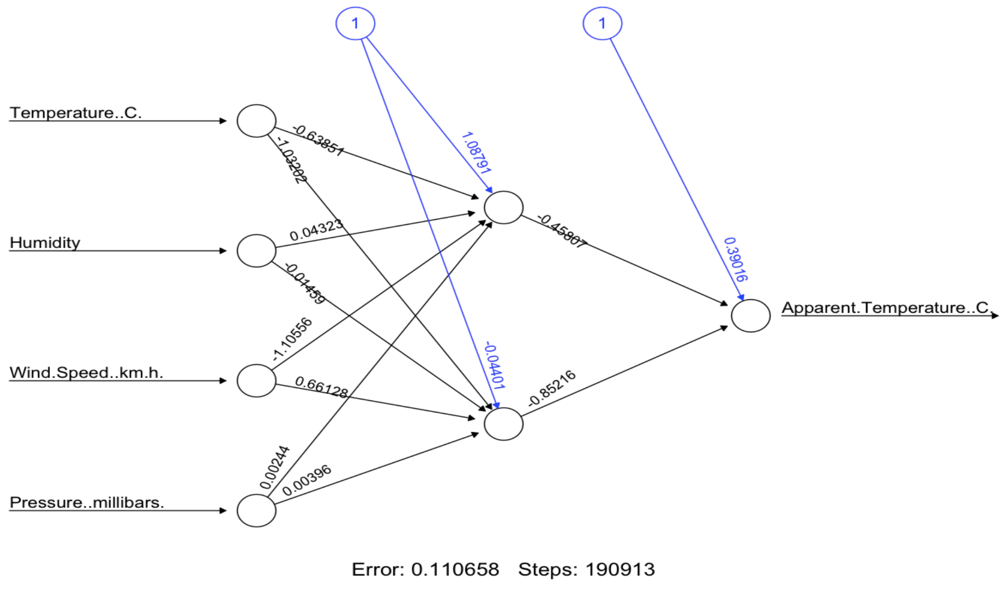
Graph B

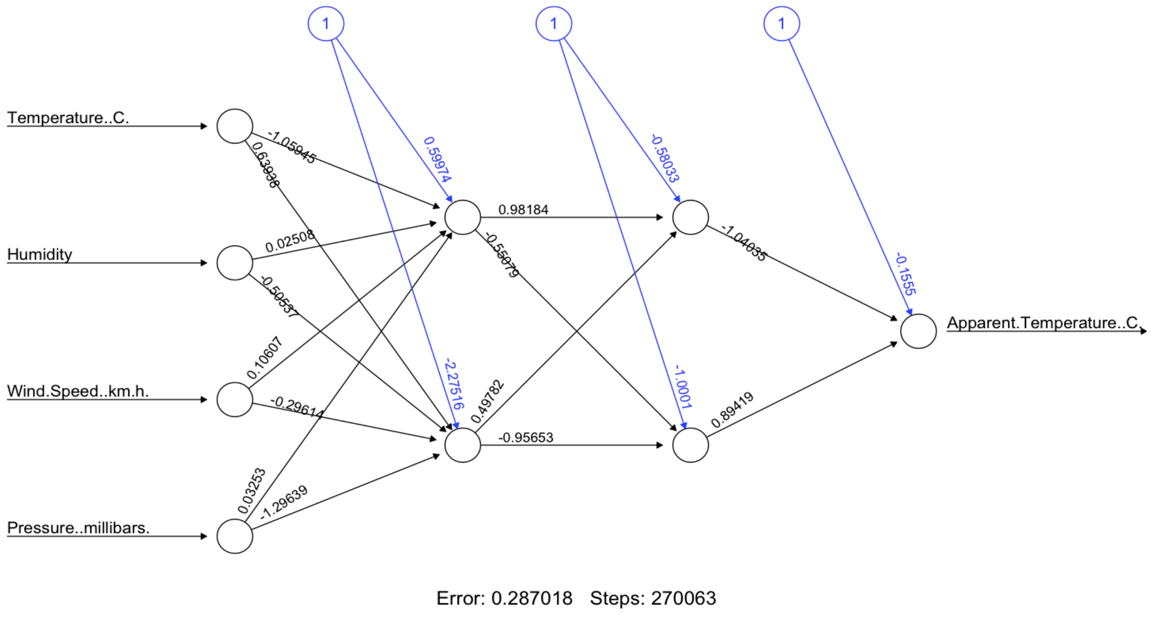


Graph A shows that predicted values missed the actual values by a huge margin when Temperature was not included among the independent variables. Predictions were a lot better in Graph B where Temperature is present among the independent variables.

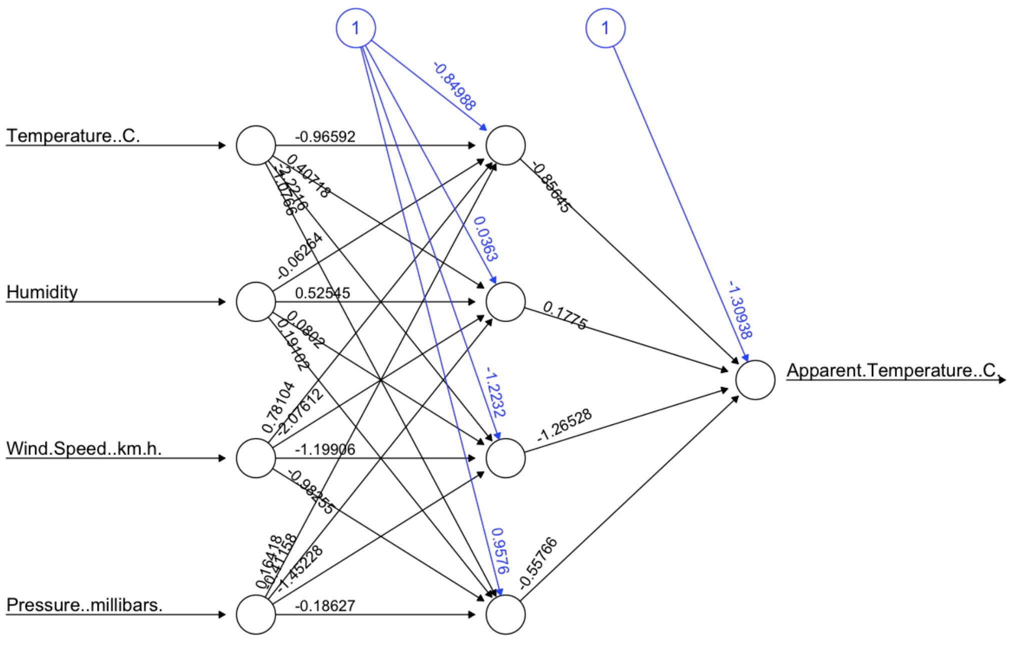
In order to optimize the error rate even more and make more accurate predictions, I started tuning various parameters of the neural network training process.

Number of hidden layers (1 hidden layer, 2 hidden layers)

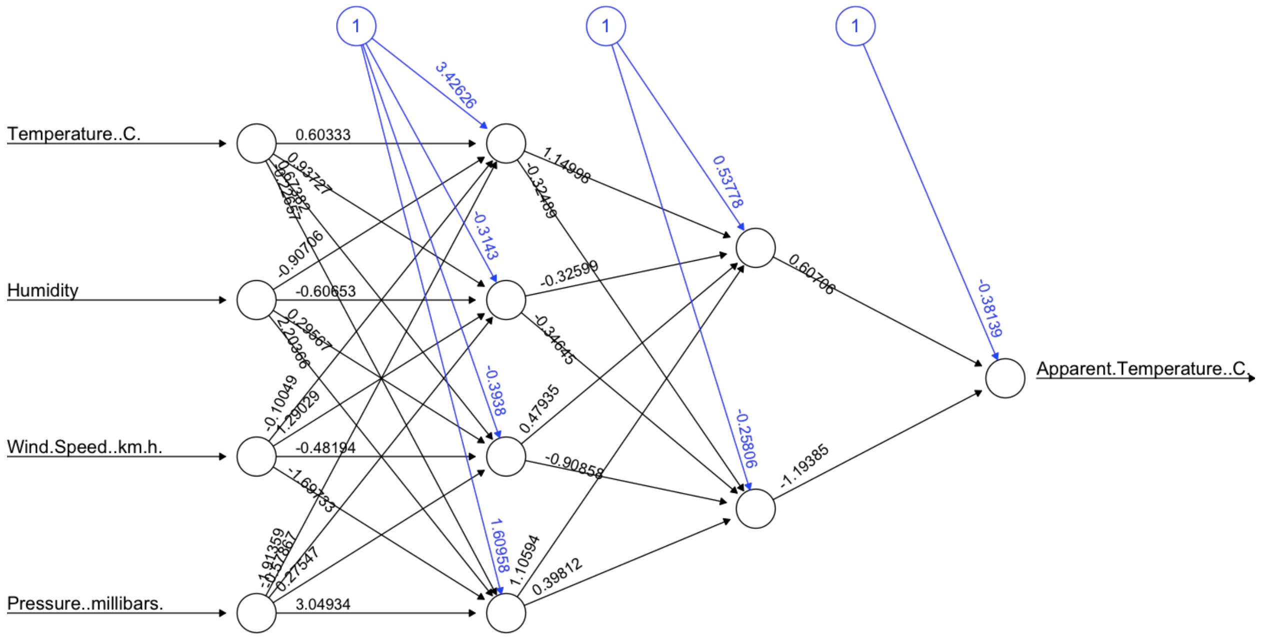




Number of neurons in each layer (One hidden layer with 4 neurons, 2 hidden layers with 4 and 2 neurons respectively)



Error: 0.07805 Steps: 322632

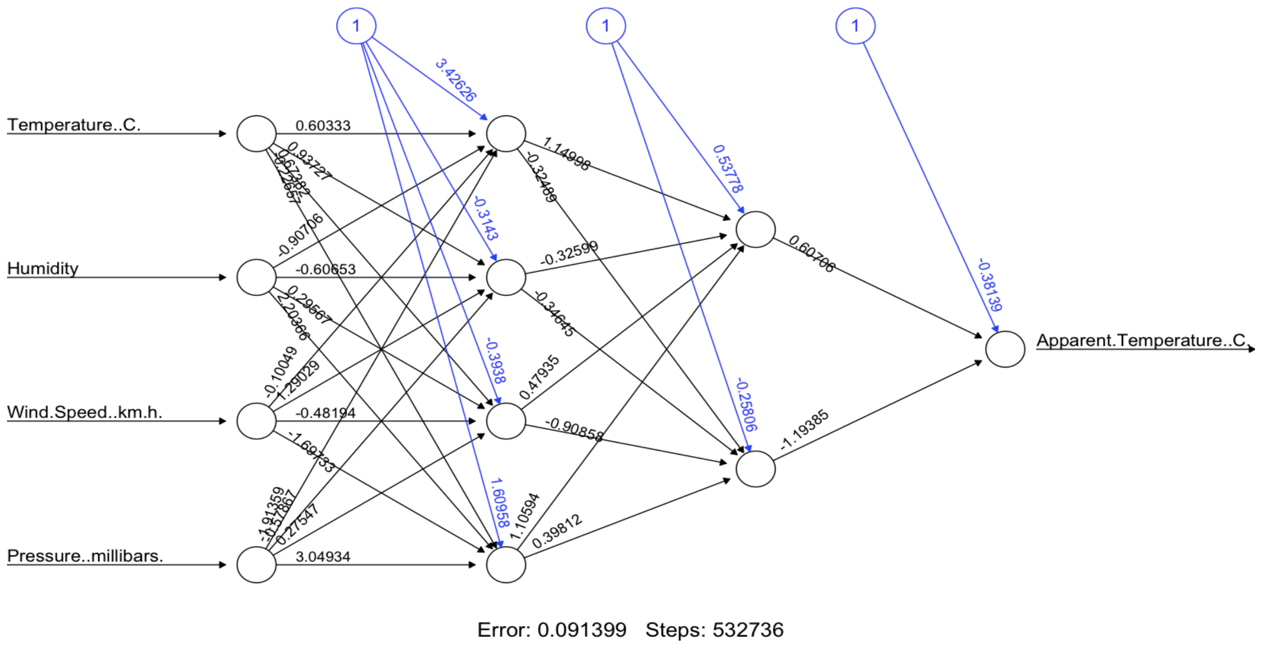


Error: 0.091399 Steps: 532736

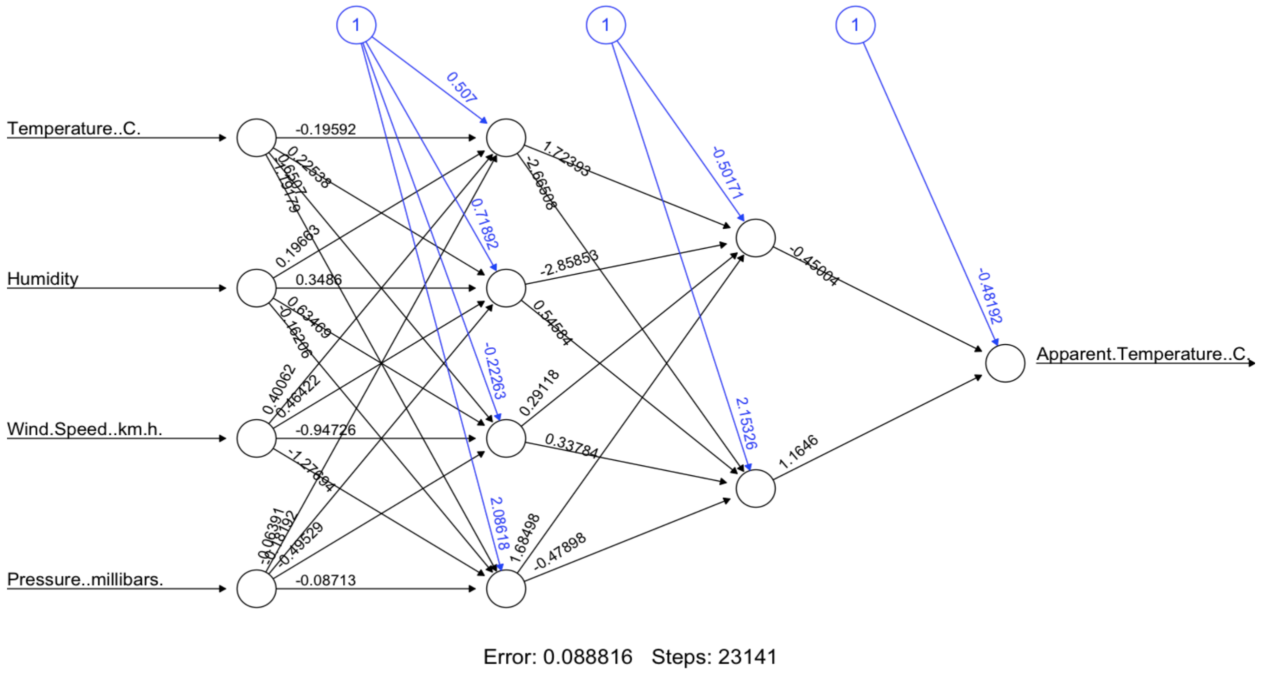
Backpropagation Algorithm (Normal Backpropagation, Resilient Backpropagation)

Resilient Backpropagation performs slightly better as compared to Normal Backpropagation

Normal

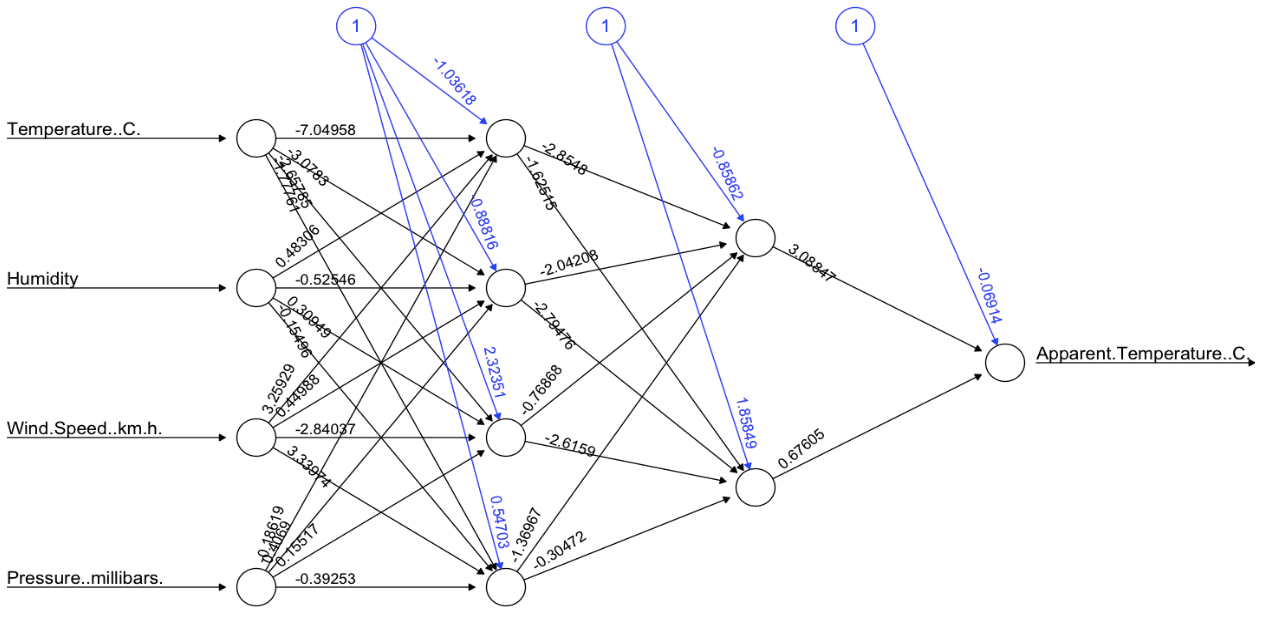


Resilient



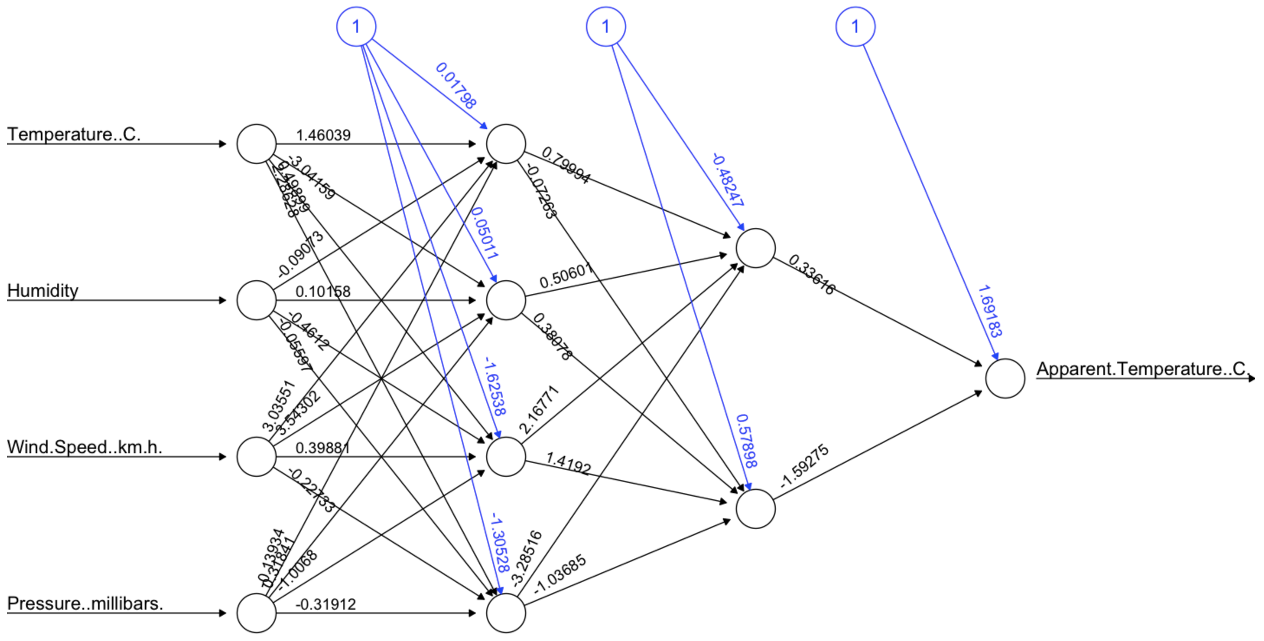
Activation Function (Sigmoid, tanh, relu)

Sigmoid



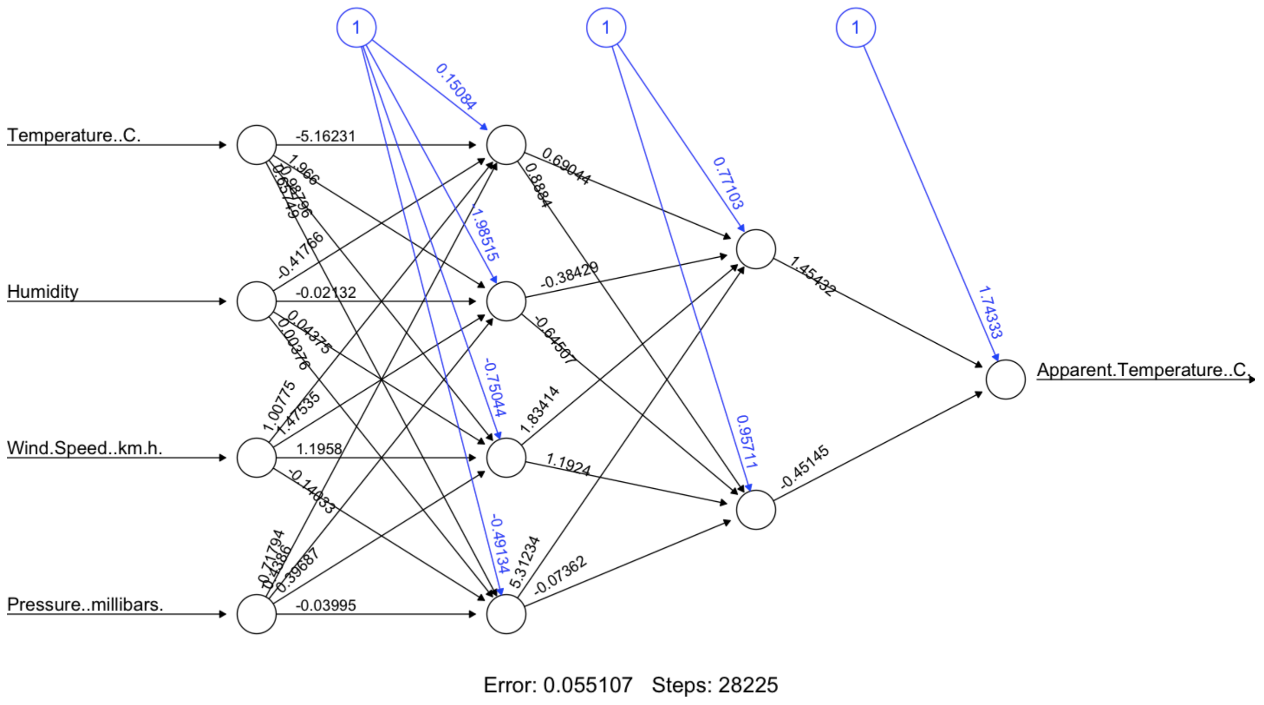
Error: 0.077815 Steps: 5184

relu



Error: 0.082333 Steps: 6409

tanh



After tuning all these parameters, lowest possible value of error rate achieved was 0.055107. Details of final network architecture is as follows:

Number of hidden layers and neurons in each layer
2 hidden layers with 4 and 2 neurons in first and second hidden layers respectively.

Backpropagation Method
Resilient Backpropagation

Activation Function
Tanh

Mean Squared Error (MSE) = 0.0000807523271

The smaller the mean squared error, the closer you are to optimizing the algorithm. It means the algorithm which gives us the smallest value of MSE, will be our algorithm of choice.

Accuracy comparison

Algorithm	**MSE**
Linear Regression	10.899382981520409
Support Vector Regression	11.979722416103748
Decision Tree Analysis	12.157561528442072
Feed Forward Neural Network	0.0000807523271

Since *Mean Square Error* is way less for Feed Forward Neural Network, it implies that Feed Forward Neural Network, for this particular problem statement and dataset, is a better algorithm than others to model the data at hand and make more accurate predictions.

Code in R

```r
install.packages('dplyr') # Only have to run once.
install.packages("neuralnet")
install.packages("ggplot2")
require(neuralnet)
library(dplyr)
library(reshape)
library(MASS)
require(ggplot2)

?neuralnet

DataFrame <- read.csv ("/Users/muhbandtekamshuru/Desktop/wireless/NN/Weather_2500_4x_1y.csv")

# add this line to remove non-numeric columns.
DataFrame <- DataFrame[, sapply(DataFrame, is.numeric)]

# find Correlation Co-efficients between the variables
d<- DataFrame[,c("Temperature..C.","Humidity","Wind.Speed..km.h.", "Pressure..millibars.", "Apparent.Temperature..C.")]
d_correlation <- as.matrix(cor(d))

d_correlation

```

```r
# find min max and scale
maxValue <- apply(DataFrame, 2, max)
minValue <- apply(DataFrame, 2, min)
DataFrame <- as.data.frame(scale(DataFrame, center = minValue, scale = maxValue-minValue))

## train, validation and test data
## train on 1500 rows
## validate on 500
## test on remaining 500 rows
ind <- sample(1:nrow(DataFrame), 1500)
trainDF <- DataFrame[ind,]
remainingInd <- c(1:2499)[-ind]
validInd <- sample(remainingInd, 500)
validDF <- DataFrame[validInd,]
testInd <- setdiff(remainingInd, validInd)
testDF <- DataFrame[testInd,]

allVars <- colnames(DataFrame)
predictorVars <- allVars[!allVars%in%"Apparent.Temperature..C."]
predictorVars <- paste(predictorVars, collapse = "+")
form=as.formula(paste("Apparent.Temperature..C.~",predictorVars, collapse = "+"))

sigmoid = function(x) {
  1 / (1 + exp(-x))
}

relu = function(x) {log(1+exp(x))}

neuralModel <- neuralnet(formula = form, hidden = c(4,2), linear.output = T, data = trainDF, stepmax=1e7,algorithm = "rprop+", act.fct = "tanh")

#All Observations
# observation1 neuralModel <- neuralnet(formula = form, hidden = c(2), linear.output = T, data = trainDF, stepmax=1e7, algorithm = 'backprop', learningrate = 0.0001, act.fct = "tanh")
# observation1 neuralModel <- neuralnet(formula = form, hidden = c(2,2), linear.output = T, data = trainDF, stepmax=1e7, algorithm = 'backprop', learningrate = 0.0001, act.fct = "tanh")
# observation1 neuralModel <- neuralnet(formula = form, hidden = c(4), linear.output = T, data = trainDF, stepmax=1e7, algorithm = 'backprop', learningrate = 0.0001, act.fct = "tanh")
# observation1 neuralModel <- neuralnet(formula = form, hidden = c(4,2), linear.output = T, data = trainDF, stepmax=1e7, algorithm = 'backprop', learningrate = 0.0001, act.fct = "tanh")
# observation2 neuralModel <- neuralnet(formula = form, hidden = c(4,2), linear.output = T, data = trainDF, stepmax=1e7, algorithm = "rprop+, act.fct = "tanh")
# observation2 neuralModel <- neuralnet(formula = form, hidden = c(4,2), linear.output = T, data = trainDF, stepmax=1e7, algorithm = "rprop+, act.fct = sigmoid)
# observation3 neuralModel <- neuralnet(formula = form, hidden = c(4,2), linear.output = T, data = trainDF, stepmax=1e7, algorithm = "rprop+, act.fct = relu)

plot(neuralModel)

## predict for test data
predictions <- neuralnet::compute(neuralModel, validDF[, 0:4])
str(predictions)

predictions <- predictions$net.result*(max(validDF$Apparent.Temperature..C.)-min(validDF$Apparent.Temperature..C))+min(validDF$Apparent.Temperature..C)
```

```r
actualValues <- (validDF$Apparent.Temperature..C)*(max(validDF$Apparent.Temperature..C) - min(validDF$Apparent.Temperature..C)) + min(validDF$Apparent.Temperature..C)

# Calculate MSE
MSE <- sum((predictions - actualValues)^2)/nrow(validDF)
MSE

#plot a graph to visualize data
ggplot() +
  geom_line(data = validDF, aes(x = validInd, y = Apparent.Temperature..C., colour = "Actual Values")) +
  geom_line(data = validDF, aes(x = validInd, y = predictions, colour = "Predicted Values"))+
  ggtitle("Actual vs Predicted when Temperature(real temperature) is present among independent attributes") +
  labs(x = "Index Values", y="Apparent Temperature")
```