Lab3 Report

Group B33

2023/12/05

Group members: Priyansh Gupta, Afshan Hashemi Hosseinabad, Ragini Gunda, Yuting Huang

Statement of contribution: Assignment 1 is mainly done by Yuting Huang, Ragini Gunda was mainly responsible for assignment 2 and, assignment 3 is mostly done by Priyansh Gupta. Results from all assignments are discussed afterwards between all of us and the group report is formulated based on the discussion. The group report is reviewed by Afshan Hashemi Hosseinabad based on the discussion.

Assignment 1: Kernel Methods

Introduction:

This experiment aims to employ kernel methods for predicting hourly temperatures at a specific date and location in Sweden. The data provided includes information about weather stations (stations.csv) and temperature measurements (temps50k.csv). The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours.

According to the requirements, we are tasked with computing three Gaussian kernels separately and utilizing the sum (final_kernel_sum) and product (final_kernel_mul) of these three kernels as the ultimate prediction kernels. Through the kernel method, we aim to obtain the final temperature predictions.

Preparation：

Clean and format the data. Select appropriate smoothing coefficient or width for each of the three kernels:

```r
# Define smoothing coefficients or widths for the three kernels
h_distance <- 30000
h_date <- 360
h_time <- 10

# Point to predict
a <- 58.4274
b <- 14.826

# Date and times for prediction
date <- as.Date("2013-5-30")
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00",
           "12:00:00", "14:00:00", "16:00:00", "18:00:00",
           "20:00:00", "22:00:00", "24:00:00")

# Data formatting and initializing
rows_to_remove <- st$date >= date
st <- st[!rows_to_remove, ]
hours <- as.POSIXct(times,format="%H:%M:%S")
st$time <- as.POSIXlt(st$time, format="%H:%M:%S")
final_kernel_sum <- length(times)
final_kernel_mul <- length(times)
df_weather <- data.frame()
```

Calculation:

Calculate the Gaussian kernels:

$$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2h^2}\right)$$

- The first kernel accounts for the physical distance from a station to the point of interest using the distHaversine function:

```r
# Calculate distances kernel
dist_diff <- distHaversine(st[, c("longitude", "latitude")],
                           c(b, a))
kernel_distance <- exp(-(dist_diff/h_distance)^2)
```

- The second kernel considers the distance between the day of a temperature measurement and the forecast day:

```r
# Calculate date kernel
date_diff <- as.numeric(difftime(as.Date(as.character(st$date)),
             date, units = "days"))
kernel_date <- exp(-(date_diff/h_date)^2)
```

- The third kernel involves the distance between the hour of a temperature measurement and the forecast hour:

```r
time_diff <- as.numeric(difftime(st$time,hours[i],
         units = "hours"))
time_diff <- abs(time_diff)
time_diff[time_diff > 12] = 24 - time_diff[time_diff > 12]
kernel_time <- exp(-(time_diff/h_time)^2)
```

- Calculate new kernels:

$$y_k(x) = \frac{\sum_n k(\frac{x-x_n}{h})t_n}{\sum_n k(\frac{x-x_n}{h})}$$

```r
for (i in seq_along(times)) {
  time_diff <- as.numeric(difftime(st$time,hours[i], units = "hours"))
  time_diff <- abs(time_diff)
  time_diff[time_diff > 12] = 24 - time_diff[time_diff > 12]
  kernel_time <- exp(-(time_diff/h_time)^2)

  # Combine kernels by summation
  combined_kernel_sum <- kernel_distance + kernel_date + kernel_time
  final_kernel_sum[i] <- sum(st$air_temperature * combined_kernel_sum) /
sum(combined_kernel_sum)

  # Combine kernels by multiplication
  combined_kernel_mul <- kernel_distance * kernel_date * kernel_time
  final_kernel_mul[i] <- sum(st$air_temperature * combined_kernel_mul) /
sum(combined_kernel_mul)
```

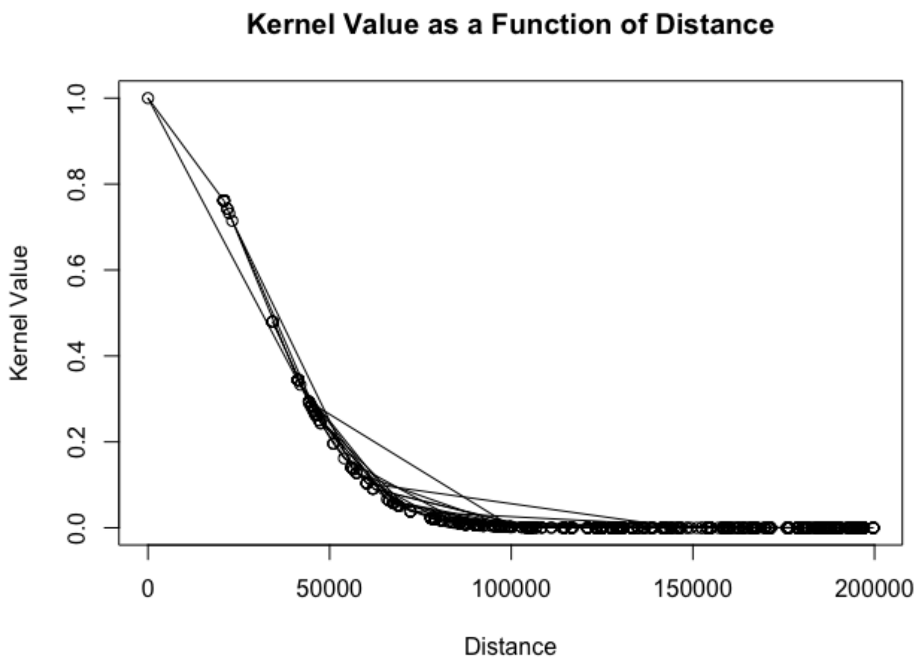
```

  df <- data.frame(TIME = times[i],
  KERNEL_SUM = final_kernel_sum[i] ,
  KERNEL_MUL = final_kernel_mul[i])
  df_weather <- rbind(df_weather, df)
}
```

Result：

TIME	KERNEL_SUM	KERNEL_MUL
04:00:00	4.003251	3.903098
06:00:00	4.206463	3.969025
08:00:00	4.461803	4.258545
10:00:00	4.691123	4.550032
12:00:00	4.853911	4.838965
14:00:00	4.907215	5.121694
16:00:00	4.842768	5.236338
18:00:00	4.671323	5.137715
20:00:00	4.393895	4.918597
22:00:00	4.120531	4.601686
24:00:00	3.916008	4.265289

Visualization:

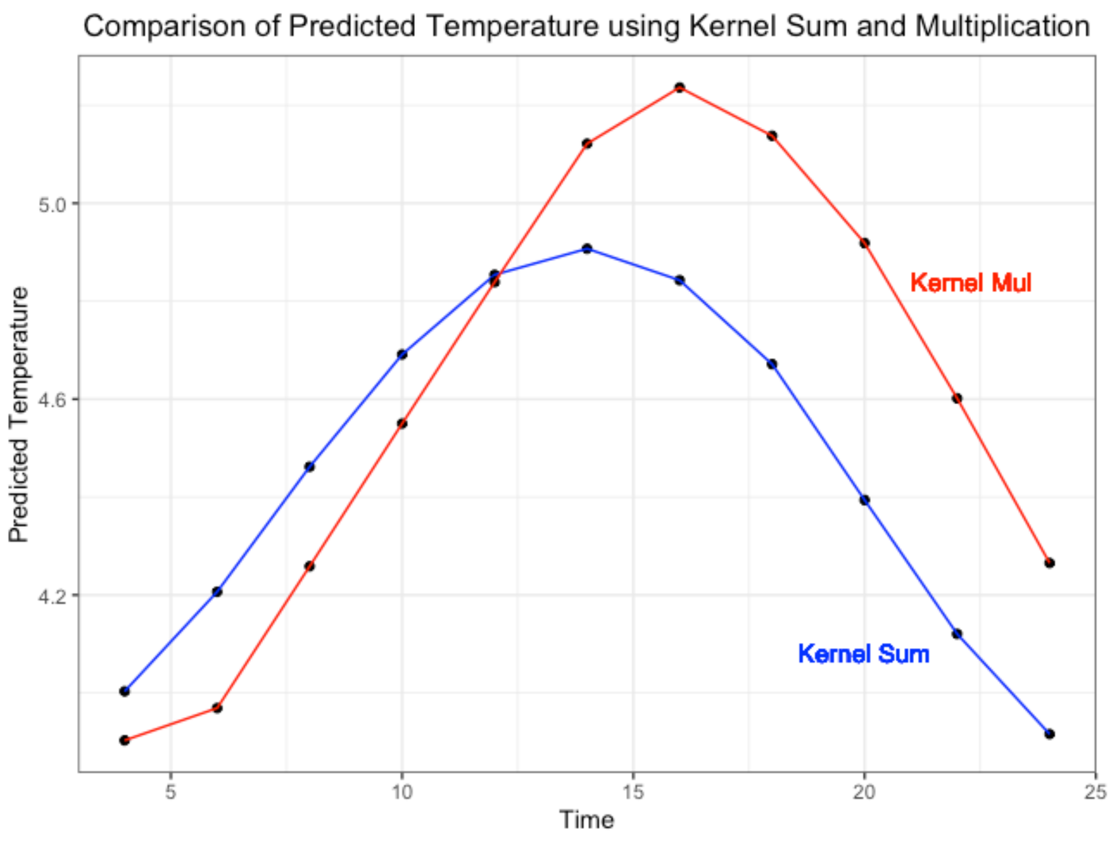
Q： Show the **plot** of the kernel value as a function of distance.



"Kernel Value as a Function of Distance" illustrates how the kernel value decreases with an increase in distance. It also helps us understand how the choice of smoothing coefficients affects the shape of the kernel.

Q： Compare the results obtained in both cases: summation of the kernels and the multiplication of kernels and elaborate on why they may differ.

```r
# Plot the combined data
ggplot(df_weather, aes(x = seq(4, 24, 2))) +
  geom_point(aes(y = final_kernel_sum), color = "black") +
  geom_line(aes(y = final_kernel_sum), color = "blue") +
  geom_point(aes(y = final_kernel_mul), color = "black") +
  geom_line(aes(y = final_kernel_mul), color = "red") +
  geom_text(aes(x = 20, y = tail(KERNEL_SUM, 1), label = "Kernel Sum"), vjust = -4,
hjust = 0.5, color = "blue") +
  geom_text(aes(x = 21, y = tail(KERNEL_MUL, 1), label = "Kernel Mul"), vjust = -15,
hjust = 0, color = "red") +
  xlab("Time") +
  ylab("Predicted Temperature") +
  ggtitle("Comparison of Predicted Temperature using Kernel Sum and Multiplication") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5), legend.position = "right")
```



Conclusion：
The difference arises from two different ways of combining the kernels (summation and multiplication), which have different impacts on the predicted results.
Kernel Sum:
When we use the three kernels combined by summation, the Gaussian kernels for physical distance, date, and time all contribute to the predicted temperature at each time point. This results in a trend in predicting temperature at different time points throughout the day. Due to the summation of kernels, at noon (12:00), all three kernels enhance the temperature, leading to a higher predicted temperature. This explains the peak in the plot.

Kernel Mul:
In this case, the contributions of the three kernels are combined by multiplication.
One characteristic of the multiplication kernel is that if any of the kernels has a value close to zero at a certain time point, the overall multiplication kernel's value tends toward zero. In this plot, at certain time points, one or more kernels have lower values, causing the overall multiplication kernel's value to be relatively low. This may result in no peak at 12:00, unlike the sum kernel plot.
Therefore, the difference in the plots is mainly due to the different ways in which the kernels' values contribute at different time points under the two kernel combination methods. This difference may stem from the fact that the Gaussian kernels are more susceptible to the influence of smaller values when multiplied.

Assignment 2: Support Vector Machines

Q1: **Which filter do you return to the user? filter0, filter1, filter2 or filter3? Why?**
I recommend the utilization of filter 2. Among the various models, filter2 demonstrates superior performance due to its extensive training on the combined dataset of trva. By employing the distinct test dataset (te), which has not been encountered during training, filter2 is better positioned for robust generalization to new, unseen data.

While the error rate of filter0 appears favorable, it is crucial to note that the hyperparameter tuning for this model is based on the training data (tr) and validation data (va). Since these datasets overlap, the seemingly lower error may be attributed to the familiarity of the model with the validation set.

On the other hand, although filter3 exhibits the lowest error rate, its training involves the entire dataset (spam), and it is evaluated on the te data, which is already present in the training data. Consequently, there is a risk of overfitting, and the model may not generalize well to new test data.

Therefore, considering the comprehensive evaluation, filter2 emerges as the recommended choice for its utilization of a diverse training set and evaluation on unseen test data, contributing to a more reliable and generalized model.

Q2: **What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why?**
The error (error1) can be considered the generalized error for filter1. This is because, while filter1 uses the training data (tr) and test data (te), the optimal value for the parameter C was obtained through a model evaluation that utilized the validation set (va). Notably, the validation set (va) was not part of the training or testing process for filter1, emphasizing the generalization aspect of the error assessment.

Q3: Once a SVM has been fitted to the training data, a new point is essentially classified according to the sign of a linear combination of the kernel function values between the support vectors and the new point. You are asked to implement this linear combination for filter3. You should make use of the functions alphaindex, coef and b that return the indexes of the support vectors, the linear coefficients for the support vectors, and the negative intercept of the linear combination. See the help file of the kernlab package for more information. You can check if your results are correct by comparing them with the output of the function predict where you set type = "decision". Do so for the first 10 points in the spam dataset. Feel free to use the template provided in the Lab3Block1 2021 SVMs St.R file.

Implementation of SVM model for filter3

```
filter3<-ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
```

The index for the support vector is given by

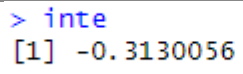
```
sv<-alphaindex(filter3)[[1]]
```

The linear coefficients for the support vectors is given by

```
co<-coef(filter3)[[1]]

```

The negative intercept if the linear combination is given by

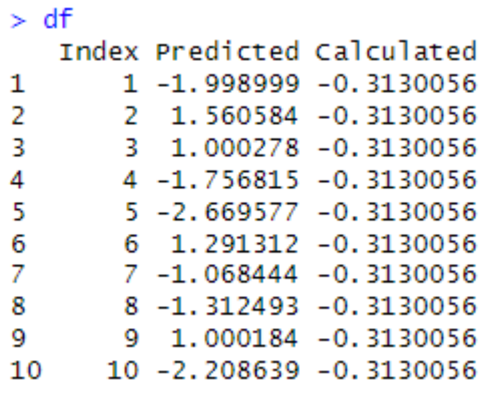
```
inte<- - b(filter3)

```



```
> inte
[1] -0.3130056
```

Comparison of results

```
sv<-alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
inte
k<-NULL

for (i in 1:10) {
  k2 <- NULL
  for (j in 1:length(sv)) {
    x1 <- as.vector(unlist(spam[i, -58]))
    x2 <- as.vector(unlist(spam[sv[j], -58]))
    # Manually calculate RBF kernel value
    k2 <- c(k2, exp(-sum((x1 - x2)^2) / (2 * filter3@kpar$sigma^2)))
  }
  # Calculate the decision value manually
  k <- c(k, sum(co * k2) + inte)
}

Predicted<-as.vector(predict(filter3,spam[1:10,-58], type="decision"))
Calculated<-k
Index<-1:10
df<-data.frame(Index,Predicted,Calculated)
df

```



```
> df
   Index Predicted Calculated
1      1 -1.998999 -0.3130056
2      2  1.560584 -0.3130056
3      3  1.000278 -0.3130056
4      4 -1.756815 -0.3130056
5      5 -2.669577 -0.3130056
6      6  1.291312 -0.3130056
7      7 -1.068444 -0.3130056
8      8 -1.312493 -0.3130056
9      9  1.000184 -0.3130056
10    10 -2.208639 -0.3130056
```

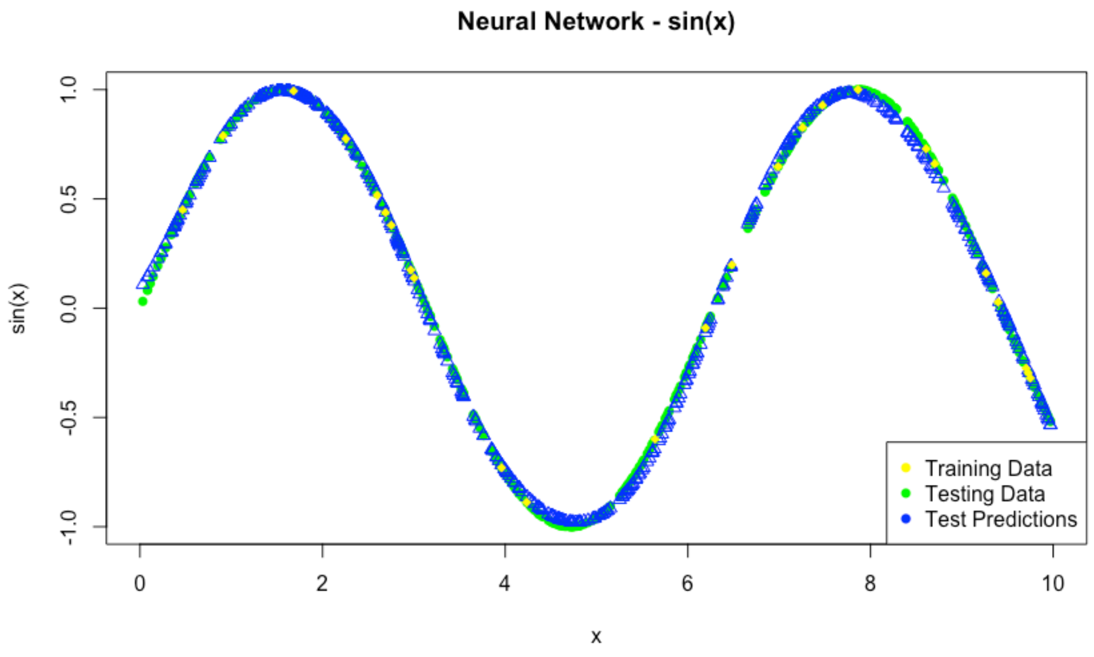
From the above dataframe, we can see that the values of Predicted and Calculated are same.

Assignment 3: Neural Network

1. Train a neural network to learn the trigonometric sine function. To do so, sample 500 points uniformly at random in the interval [0,10]. Apply the sine function to each point. The resulting value pairs are the data points available to you. Use 25 of the 500 points for training and the rest for test. Use **one hidden layer with 10 hidden units**. You do not need to apply early stopping. Plot the training and test data, and the predictions of the learned NN on the test data. You should get good results. Comment your results.

```r
# generating 500 data points between interval [0,10]
data_points <- data.frame(x=runif(500,0,10))
data_points$y <-sin(data_points$x)
# split the training and testing data
index<-sample(1:500, 25)
trainingdata<- data_points[index,]
testingdata<- data_points[-index,]
#NeuralNetwork
neuralNetwork <- neuralnet(y~x, trainingdata, hidden = c(10), linear.output = TRUE)
neuralNetwork$weights

#prediction on test data
predictTest<-predict(neuralNetwork, testingdata)

plot(testingdata$x, testingdata$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)",
main = "Neural Network - sin(x)")
points(testingdata$x, predictTest, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
legend("bottomright", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col =
c("red","green","blue"), pch = 16)

```

Yellow colour points represents the training point, Green colour in below plot represent both testing data points, whereas blue colour represents the prediction on the testing data. As we can see in the plot, model has learned the trigonometric sine function well. There are some fluctuation while predicting the sine function on test data as we can see on the plot. It is because we just use the 25 points for training. If we would have supplied more training points, we would have not seen this fluctuation also.



2. In question (1), you used the default logistic (a.k.a. sigmoid) activation function, i.e. act.fct = "logistic". Repeat question (1) with the following custom activation functions: h_1 (x) = x, h_2 (x) = max{0, x} and h_3 (x) = ln(1 + exp x) (a.k.a. linear, ReLU and softplus). See the help file of the neuralnet package to learn how to use custom activation functions. Plot and comment your results.

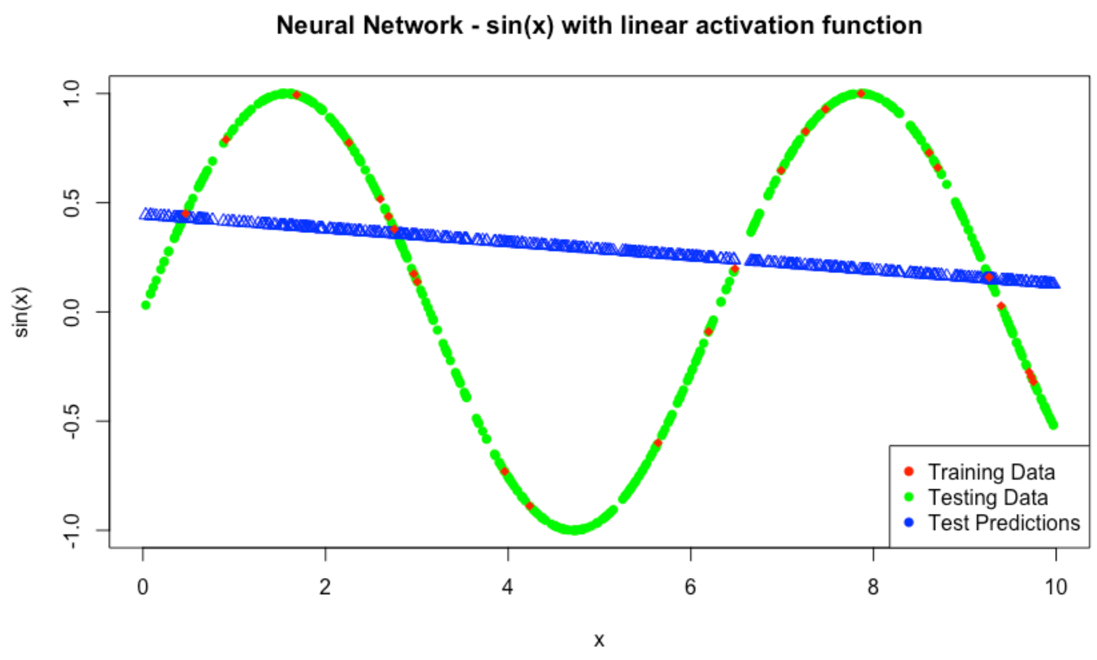
i. Linear function:

So, this question asks us to use the customer activation function instead of the default one, the first one of the three is linear function. As we know that the liner equation is represent by

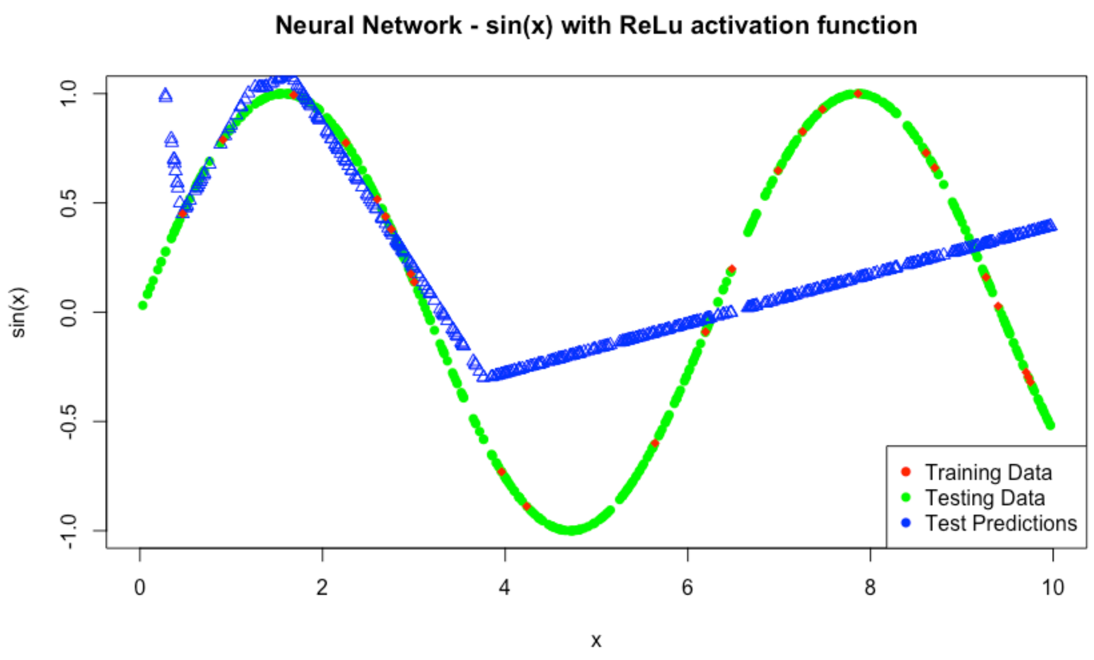
$$y = mx + c$$

so, for each value of x will give us y values. As per the property of linear function we will get a straight line when we plot it. Whereas I think this activation function should not be preferred in predicting the sine function. As actual plot of sine function is not a straight line, therefore it will not give us good prediction. Same has been shown in below code and the plot.

```
h1<-function(x) x
h2<- function (x) ifelse(x>0,x,0)
h3<- function(x) log(1 + exp(x))

# neural netweok to use above 3 functions as activation function
neuralNetwork_h1 <- neuralnet(y ~ x, data = trainingdata, hidden = c(10),linear.output =
TRUE, act.fct = h1)
neuralNetwork_h2 <- neuralnet(y ~ x, data = trainingdata, hidden = c(10), linear.output =
TRUE, act.fct = h2)
neuralNetwork_h3 <- neuralnet(y ~ x, data = trainingdata, hidden = c(10), linear.output =
TRUE, act.fct = h3)

#Predicting on traoing data
PredictTestH1<- predict(neuralNetwork_h1, testingdata)
PredictTestH2<- predict(neuralNetwork_h2, testingdata)
PredictTestH3<- predict(neuralNetwork_h3, testingdata)

#Ploting the prediction with original data
plot(testingdata$x, testingdata$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)",
main = "Neural Network - sin(x) with linear activation function")
points(testingdata$x, PredictTestH1, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
legend("bottomright", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col =
c("red","green","blue"), pch = 16)
```



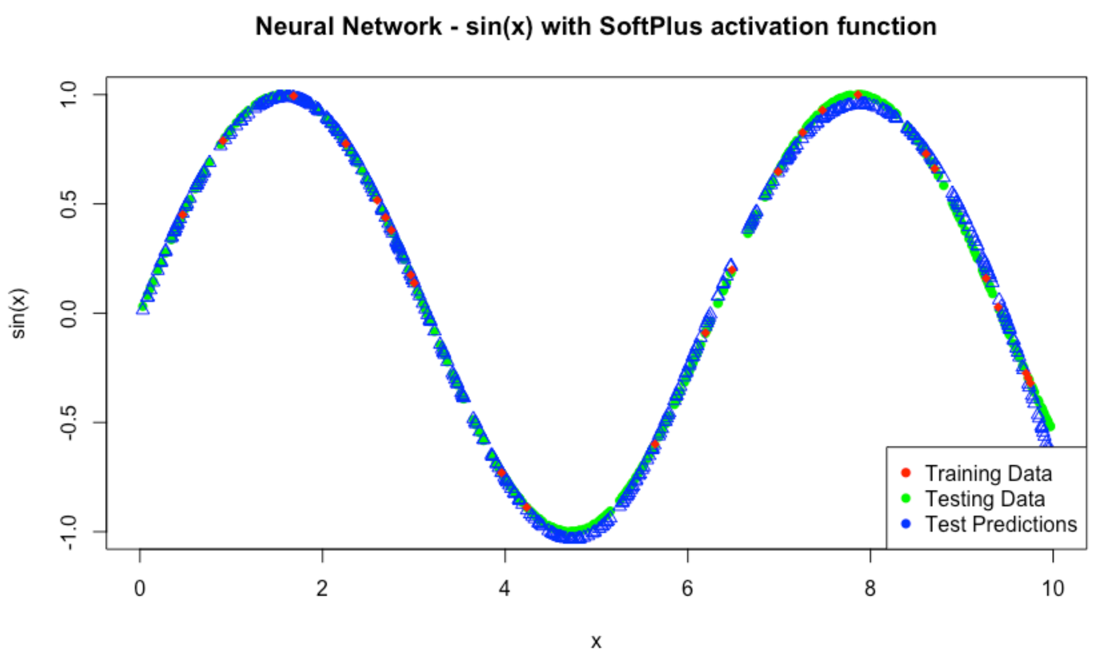
ii. ReLU function: Next we need to try the same function with ReLU function which stands for rectified linear unit. This function simply puts all the positive values to linear form. It only takes the positive values. Below is the plot for training and testing data points along with the prediction on testing data with ReLU as activation function.



As we can see in above Plot, ReLu function showed its characteristics for positive values and plot them linearly, but it deviates when it gets negative values and plot all of them in a straight line. As we can see in the plot, this activation function will also not be preferable in predicting the sine function.

iii. softplus Function: here we will be trying the same steps with softplus as activation function. SoftPlus function is defined as

$$y = ln(1 + exp\, x)$$

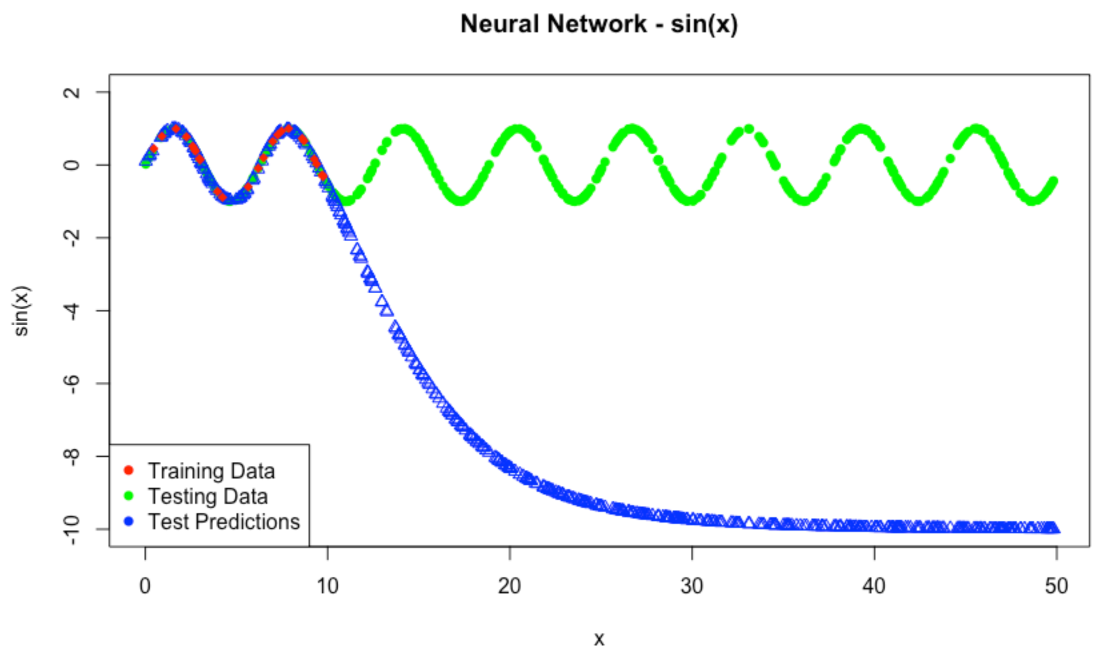


The above plot represents the use of softmax function for predicting the sine function. We can clearly see that the result of this prediction is accurate compared to previous functions. There is a slight deviation in predicting the values of maxima and minima as can be seen in the plot. As this activation function gives us the best result. We can say that softmax is preferred over both the previous function for predicting the sine function.

3. Sample 500 points uniformly at random in the interval [0,50], and apply the sine function to each point. Use the NN learned in question (1) to predict the sine function value for these new 500 points. You should get mixed results. Plot and comment your results.

```r
new_data_points <- data.frame(x=runif(500,0,50))
new_data_points$y <-sin(new_data_points$x)
predict_newdata <- predict(neuralNetwork, new_data_points)

plot(new_data_points$x, new_data_points$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)", main = "Neural Network - sin(x)", ylim=c(-10.0,2.0))
points(new_data_points$x, predict_newdata, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
legend("bottomleft", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col = c("red","green","blue"), pch = 16)

```

Below is the plot of predicting these 500 points using the NN constructed in question 1.



As we can see in the plot, The neural network perform really good for predicting the values between [0,10], whereas it deviate from actual values in interval (10,50]. The reason behind this deviation is, as we train the Network in interval [0.10] only.

4. In question (3), the predictions seem to converge to some value. Explain why this happens. To answer this question, you may need to get access to the weights of the NN learned. You can do it by running nn or nn$weights where nn is the NN learned.

Below are the weights of the neural network used in question 3.

```
>neuralNetwork$weights
[[1]]
[[1]][[1]]
          [,1]      [,2]      [,3]      [,4]      [,5]       [,6]       [,7]       [,8]
[1,]  1.683927  3.932431. 1.054123 0.8754801  4.325926 -2.6377620  0.6487447  0.3689092
[2,] -2.318679 -1.895624 -1.390587 1.3074790 -1.651400  0.2594473 -0.1308971 -1.5480956
          [,9]       [,10]
[1,] -0.5287089 -7.792404
[2,] 1.3267148  1.223525

[[1]][[2]]
             [,1]
 [1,]   0.6553536
 [2,]  -1.2009800
 [3,]  -1.8168821
 [4,]   1.6960766
 [5,]  -0.7454437
 [6,]   2.5090915
 [7,] -15.6032591
 [8,]   4.5407554
 [9,]  -4.3237190
[10,]  -0.9937610
[11,]   6.6807956
```

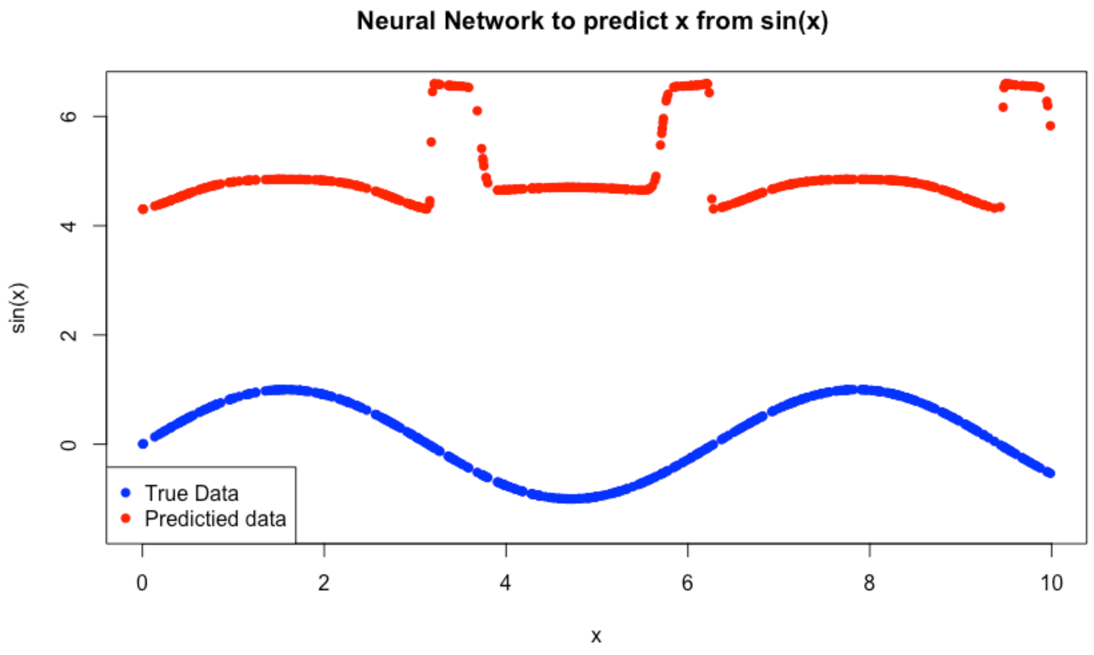
The prediction value in above plot tries to converge to -10, That is because of the weights specially in output layer, the weight from 7th unit from hidden layer to output is -15, which is pulling the value in negative direction. The values tries to converge because of optimization algorithm which tries to minimize the error between true and predicted value. The default optimization algorithm used by neuralnet function is resilient backpropagation (Rprop +). Which adjust the weight of the NN in the direction of steepest descent.

5. Sample 500 points uniformly at random in the interval [0,10], and apply the sine function to each point. Use all these points as training points for learning a NN that tries to predict x from sin(x), i.e. unlike before when the goal was to predict sin(x) from x. Use the learned NN to predict the **training data**. You should get bad results. Plot and comment your results. **Help**: Some people get a convergence error in this question. It can be solved by stopping the training before reaching convergence by setting threshold = 0.1.

Below is the code for predicting x from sin(x).

```
#part 5
data_points_part5 <- data.frame(x=runif(500,0,10))
data_points_part5$y <-sin(data_points_part5$x)

#training a neural network for predicting the sin(x) from x
nn_y_to_x <-neuralnet(x~y, data = data_points_part5, hidden=c(10), linear.output = TRUE,
threshold = 0.1 )

#predicting the training data fron NN
predictx <- predict(nn_y_to_x,data_points_part5 )

#ploting the result
plot(data_points_part5$x, data_points_part5$y, col = "blue", pch = 16, xlab = "x", ylab =
"sin(x)", main = "Neural Network to predict x from sin(x) ", ylim=c(-1.5,6.5))
points(data_points_part5$x, predictx, col = "red", pch = 16)
legend("bottomleft", legend = c("True Data", "Predictied data"), col = c("blue", "red"), pch
= 16)
```

Below plots shows the prediction of the x from sine function. As we can see from the plot predicting the value from sine function of the particular value doesn't show a very good result. We have tried with different activation function still the result is bad. Therefore, we can say that predicting any function the values is easy and we can get good result in that by changing the activation function, whereas predicting the value from a function is difficult and doesn't show good result even after changing the activation functions.



Appendix:
Code for Assignment 1:

```r
set.seed(1234567890)
library(geosphere)
library(ggplot2)

# Load data
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations, temps, by = "station_number")


# Define smoothing coefficients or widths for the three kernels
h_distance <- 30000
h_date <- 360
h_time <- 10

# Point to predict
a <- 58.4274
b <- 14.826

# Date and times for prediction
date <- as.Date("2013-5-30")
times  <-  c("04:00:00",  "06:00:00",  "08:00:00",  "10:00:00",  "12:00:00",  "14:00:00",
"16:00:00", "18:00:00", "20:00:00", "22:00:00", "24:00:00")

# Data formatting and initializing
rows_to_remove <- st$date >= date
st <- st[!rows_to_remove, ]
hours <- as.POSIXct(times,format="%H:%M:%S")
st$time <- as.POSIXlt(st$time, format="%H:%M:%S")
final_kernel_sum <- length(times)
final_kernel_mul <- length(times)
```

```r
df_weather <- data.frame()

# Calculate distances kernel
dist_diff <- distHaversine(st[, c("longitude", "latitude")], c(b, a))
kernel_distance <- exp(-(dist_diff/h_distance)^2)

# Calculate date kernel
date_diff <- as.numeric(difftime(as.Date(as.character(st$date)), date, units = "days"))
kernel_date <- exp(-(date_diff/h_date)^2)

# Calculate time kernel
for (i in seq_along(times)) {
  time_diff <- as.numeric(difftime(st$time,hours[i], units = "hours"))
  time_diff <- abs(time_diff)
  time_diff[time_diff > 12] = 24 - time_diff[time_diff > 12]
  kernel_time <- exp(-(time_diff/h_time)^2)

  # Combine kernels by summation
  combined_kernel_sum <- kernel_distance + kernel_date + kernel_time
  final_kernel_sum[i] <- sum(st$air_temperature * combined_kernel_sum) / sum(combined_kernel_sum)

  # Combine kernels by multiplication
  combined_kernel_mul <- kernel_distance * kernel_date * kernel_time
  final_kernel_mul[i] <- sum(st$air_temperature * combined_kernel_mul) / sum(combined_kernel_mul)

  df <- data.frame(TIME = times[i], KERNEL_SUM = final_kernel_sum[i], KERNEL_MUL = final_kernel_mul[i])
  df_weather <- rbind(df_weather, df)
}

knitr::kable(df_weather)

#Plot of the kernel value as a function of distance
filtered_dist_diff <- dist_diff[dist_diff < 200000]
filtered_kernel_distance <- kernel_distance[dist_diff < 200000]
num_samples <- 1000
sample_indices <- seq(1, length(filtered_dist_diff), length.out = num_samples)
sampled_dist_diff <- filtered_dist_diff[sample_indices]
sampled_kernel_distance <- filtered_kernel_distance[sample_indices]

# Plot the kernel value as a function of distance
plot(sampled_dist_diff, sampled_kernel_distance, type = "o",
     main = "Kernel Value as a Function of Distance",
     xlab = "Distance", ylab = "Kernel Value")


# Plot the combined data
ggplot(df_weather, aes(x = seq(4, 24, 2))) +
  geom_point(aes(y = final_kernel_sum), color = "black") +
  geom_line(aes(y = final_kernel_sum), color = "blue") +
  geom_point(aes(y = final_kernel_mul), color = "black") +
  geom_line(aes(y = final_kernel_mul), color = "red") +
  geom_text(aes(x = 20, y = tail(KERNEL_SUM, 1), label = "Kernel Sum"), vjust = -4, hjust = 0.5, color = "blue") +
  geom_text(aes(x = 21, y = tail(KERNEL_MUL, 1), label = "Kernel Mul"), vjust = -15, hjust = 0, color = "red") +
  xlab("Time") +
  ylab("Predicted Temperature") +
  ggtitle("Comparison of Predicted Temperature using Kernel Sum and Multiplication") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5), legend.position = "right")
```

Code for assignment 2:

```r
#install.packages("kernlab")
library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58])
tr <- spam[1:3000, ]
va <- spam[3001:3800, ]
trva <- spam[1:3800, ]
te <- spam[3801:4601, ]

by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=i,scaled=FALSE)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}

filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0

filter1 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter1,te[,-58])
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
err1

filter2 <- ksvm(type~.,data=trva,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter2,te[,-58])
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
err2

filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3
sv<-alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
inte
k<-NULL

for (i in 1:10) {
  k2 <- NULL
  for (j in 1:length(sv)) {
      x1 <- as.vector(unlist(spam[i, -58]))
      x2 <- as.vector(unlist(spam[sv[j], -58]))
      # Manually calculate RBF kernel value
      k2 <- c(k2, exp(-sum((x1 - x2)^2) / (2 * filter3@kpar$sigma^2)))
    }
    # Calculate the decision value manually
    k <- c(k, sum(co * k2) + inte)
  }

Predicted<-as.vector(predict(filter3,spam[1:10,-58], type="decision"))
Calculated<-k
Index<-1:10
df<-data.frame(Index,Predicted,Calculated)
df

```

Code for assignment 3:

```r
#install.packages("neuralnet")
library(neuralnet)

set.seed(1234567890)
#part-1
# generating 500 data points between interval [0,10]
data_points <- data.frame(x=runif(500,0,10))
data_points$y <-sin(data_points$x)
# split the training and testing data
index<-sample(1:500, 25)
trainingdata<- data_points[index,]
testingdata<- data_points[-index,]
#NeuralNetwork
neuralNetwork <- neuralnet(y~x, trainingdata, hidden = c(10), linear.output = TRUE)
neuralNetwork$weights

#prediction on test data
predictTest<-predict(neuralNetwork, testingdata)

plot(testingdata$x, testingdata$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)", main = "Neural Network - sin(x)")
points(testingdata$x, predictTest, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
legend("bottomright", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col = c("red","green","blue"), pch = 16)

#part-2
h1<-function(x) x
h2<- function (x) ifelse(x>0,x,0)
h3<- function(x) log(1 + exp(x))

# neural netweok to use above 3 functions as activation function
neuralNetwork_h1 <- neuralnet(y ~ x, data = trainingdata, hidden = c(10),linear.output = TRUE, act.fct = h1)
neuralNetwork_h2 <- neuralnet(y ~ x, data = trainingdata, hidden = c(10), linear.output = TRUE, act.fct = h2)
neuralNetwork_h3 <- neuralnet(y ~ x, data = trainingdata, hidden = c(10), linear.output = TRUE, act.fct = h3)

#Predicting on traoing data
PredictTestH1<- predict(neuralNetwork_h1, testingdata)
PredictTestH2<- predict(neuralNetwork_h2, testingdata)
PredictTestH3<- predict(neuralNetwork_h3, testingdata)

#Ploting the prediction with original data
plot(testingdata$x, testingdata$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)", main = "Neural Network - sin(x) with linear activation function")
points(testingdata$x, PredictTestH1, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
```

```r
legend("bottomright", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col = c("red","green","blue"), pch = 16)

plot(testingdata$x, testingdata$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)", main = "Neural Network - sin(x) with ReLu activation function")
points(testingdata$x, PredictTestH2, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
legend("bottomright", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col = c("red","green","blue"), pch = 16)

plot(testingdata$x, testingdata$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)", main = "Neural Network - sin(x) with SoftPlus activation function")
points(testingdata$x, PredictTestH3, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
legend("bottomright", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col = c("red","green","blue"), pch = 16)

#part 3
new_data_points <- data.frame(x=runif(500,0,50))
new_data_points$y <-sin(new_data_points$x)
predict_newdata <- predict(neuralNetwork, new_data_points)

plot(new_data_points$x, new_data_points$y, col = "green", pch = 16, xlab = "x", ylab = "sin(x)", main = "Neural Network - sin(x)", ylim=c(-10.0,2.0))
points(new_data_points$x, predict_newdata, col = "blue", pch = 2)
points(trainingdata$x, trainingdata$y,col="red", pch=18)
legend("bottomleft", legend = c("Training Data" ,"Testing Data", "Test Predictions"), col = c("red","green","blue"), pch = 16)

#part4
neuralNetwork$weights

#part 5
data_points_part5 <- data.frame(x=runif(500,0,10))
data_points_part5$y <-sin(data_points_part5$x)

#training a neural network for predicting the sin(x) from x
nn_y_to_x <-neuralnet(x~y, data = data_points_part5, hidden=c(10), linear.output = TRUE, threshold = 0.1 )

#predicting the training data fron NN
predictx <- predict(nn_y_to_x,data_points_part5 )

#ploting the result
plot(data_points_part5$x, data_points_part5$y, col = "blue", pch = 16, xlab = "x", ylab = "sin(x)", main = "Neural Network to predict x from sin(x) ", ylim=c(-1.5,6.5))
points(data_points_part5$x, predictx, col = "red", pch = 16)
legend("bottomleft", legend = c("True Data", "Predictied data"), col = c("blue", "red"), pch = 16)


```