Machine Learning Computer Lab 1 Block2 (Group A7)

Qinyuan Qi(qinqi464) Satya Sai Naga Jaya Koushik Pilla (satpi345)
Daniele Bozzoli(danbo826)

2024-01-14

Statement of Contribution

For Lab 3, we decided to split the three assignments equally, Qinyuan completed Assignment 1, Satya completed Assignment 2 and Daniele completed Assignment 3, after which, for verification's sake, we completed each other's assignments as well and validated our findings. The report was also compiled by three of us, with each handling their respective assignments.

Assignment 1: KERNEL METHODS(Solved by Qinyuan Qi)

Answer:

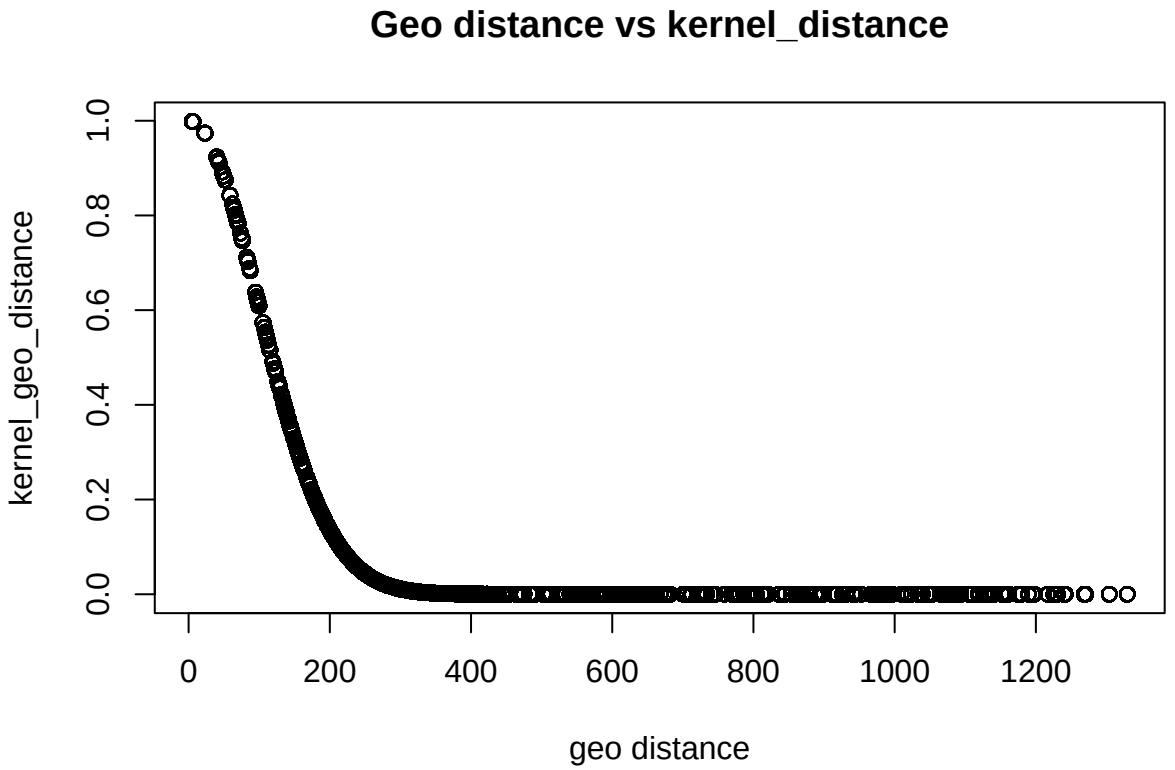
In this assignment, we will try to predict the temperature of Linköping University Valla Campus on the date 2010-1-1.

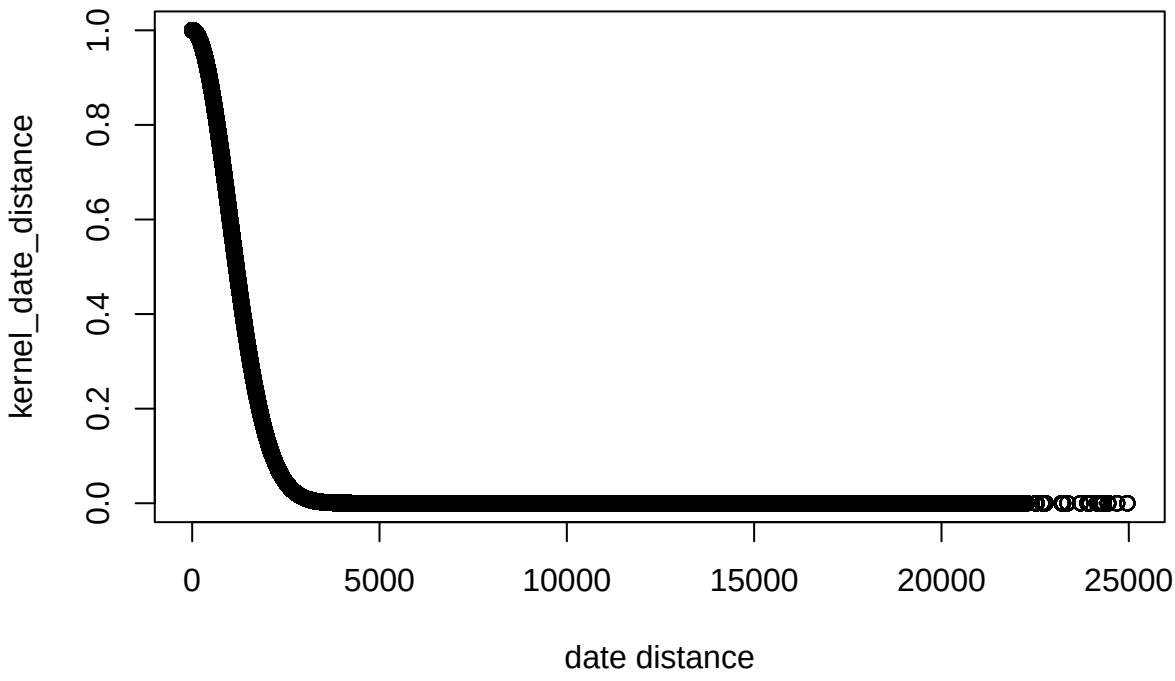
We set the following parameters which will be used in our kernel functions.

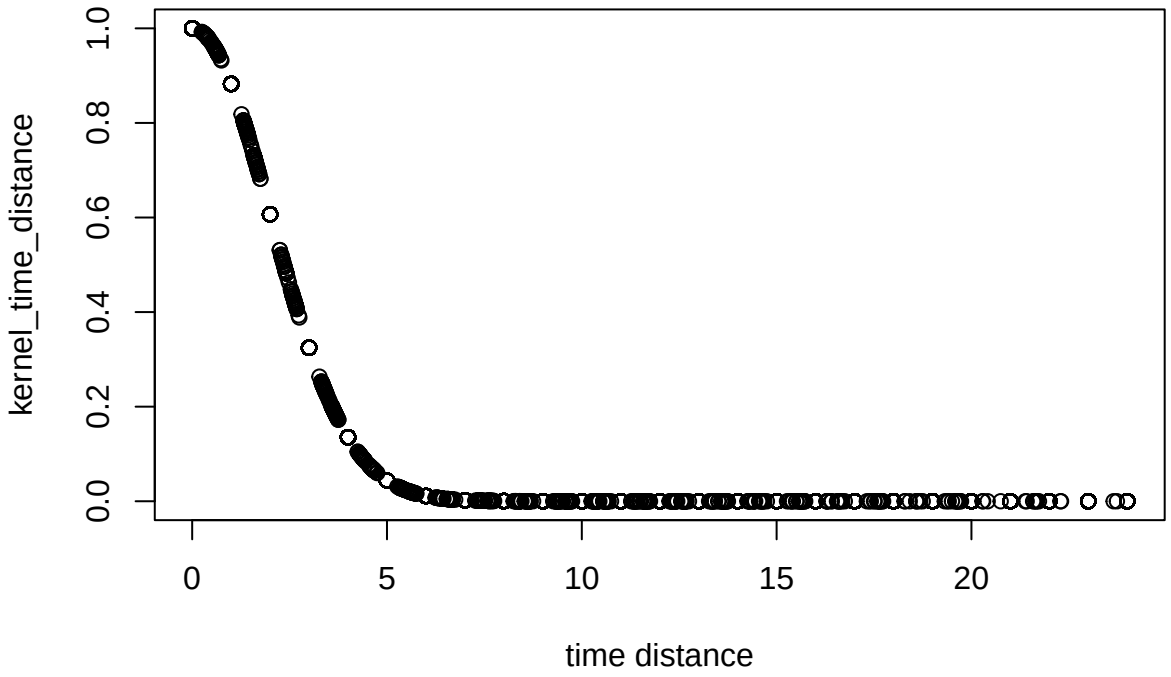
```
# define the date of interest
date_interest <- as.Date("2010-1-1")

# Point to predict(Linköping University Valla Campus Geo Location)
a <- 58.3986
b <- 15.5780

# other parameters
# distance unit is KM
h_distance <- 100
h_date <- 1000
h_time <- 2
# It will generate a sequence of time from 4:00:00 to 24:00:00 with a step of 2 hours
times_interest <- c(paste0("0",seq(4,8,by=2),":00:00"), paste0(seq(10,24,by=2),":00:00"))
```


Geo distance vs kernel_distance
kernel_geo_distance
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000
1200
geo distance


date distance vs kernel_date_distance
kernel_date_distance
0.0
0.2
0.4
0.6
0.8
1.0
0
5000
10000
15000
20000
25000
date distance

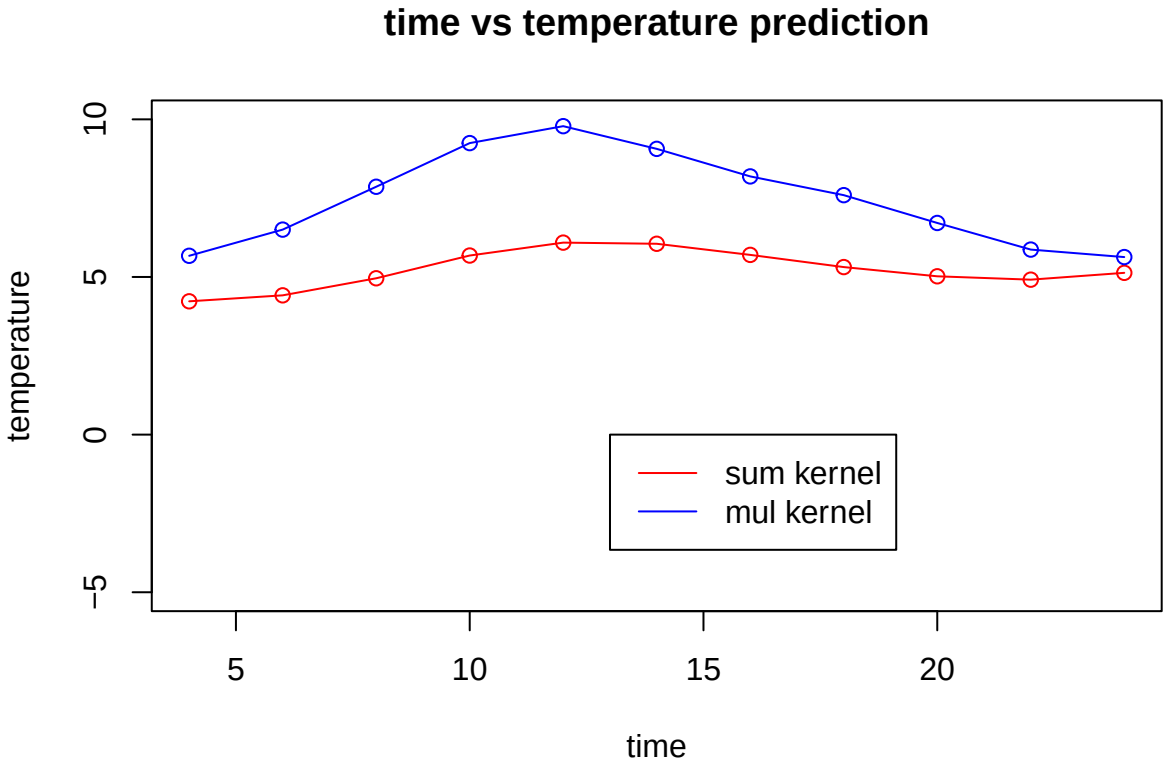



```
## The predicted temperature of the sum kernel:

##  [1] 4.228298 4.417323 4.958978 5.680031 6.089854 6.053219 5.701034 5.314121
##  [9] 5.021299 4.915237 5.131740

## The predicted temperature of the mul kernel:

##  [1] 5.672180 6.503036 7.861849 9.246307 9.784012 9.063767 8.190361 7.595429
##  [9] 6.712912 5.868893 5.631465
```



Comparing the sum and mul versions of the kernel, we can see that there is around a 1~5 degree difference. mul version has a higher temperature than the sum version.

According to the History Web Data, we can find that the real temperature of Linköping on the date 2010-1-1(Monday) is 0 to -3 °C. And we checked the temperature of Linköping that week, after that day, the temperature went down gradually, and reached -19 °C on 2010-1-6(Saturday).

Then we checked the data in the temps50k.csv, and only found 19 records of the temperature of Linköping station(station id: 85250) which is not enough to predict the temperature of Linköping on the date 2010-1-1. This is the reason why the gap between the prediction and the real temperature existed.

According to what we found, we cannot decide which version of the kernel is better, but both of them in an acceptable range.

Assignment 2: SUPPORT VECTOR MACHINES(Solved by Satya Sai Naga Jaya Koushik Pilla)

Answer:

2.1. Which filter do we return to the user ? filter0, filter1, filter2 or filter3? Why?

After checking the code, we know that the spam data is split into training, validation, training+validation, and test data respectively.

The error rate rates of 4 filters are as follows.

```
## error rate of filter0 is 6.75 %
## error rate of filter1 is 8.489388 %
## error rate of filter2 is 8.2397 %
## error rate of filter3 is 2.122347 %
```

Here, Filter 0 and Filter 1 are almost the same, both of them training on the training data, the only difference is Filter 0 predicted on validation data, and Filter 1 predicted on test data.

While Filter2 training on training+validation data predicated on test data, and Filter3 training on all data, and predicted on test data.

Regarding which filter we should return to the user. I think it should be Filter 1. Or we can use Filter 0 since both validation and test data are not touched by the model which can be swapped.

In Filter 1, we train the model using training data and then adjust the hyperparameter(C) using validation data, and predict on test data which is a best practice in the machine learning field.

The problem with Filter 2 is it trains the model on training+validation data, which makes it impossible to find new data to adjust the hyperparameter(C).

The problem with Filter 3 is it trains the model on all data, which means the test data is touched by the training process.

Since C is calculated at first which is based on the training data, to use filter 2, we need to recalculate C based on training+validation data.

```
## C based on Tr + Va is  3.9
```

```
## C based on TrVa + Te is 4.5
```

2.2 What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why?

Generalization error is the expectation of the predicted values of an independent test set on our model.

```
## generalization error of filter0 is 8.489388 %
```

```
## generalization error of filter1 is 8.489388 %
```

```
## generalization error of filter2 is 8.2397 %
```

```
## generalization error of filter3 is 2.122347 %
```

Same as what has been discussed in 2.1, err1 should be returned to the user which means we trained using training data, found C using validation data, and predicted on test data.

2.3 Implementation of SVM predictions.

Please check the appendix for the implementation of the missing code.

```
## Calculated Prediction is:
```

```
##  [1] -1.998999  1.560584  1.000278 -1.756815 -2.669577  1.291312 -1.068444
##  [8] -1.312493  1.000184 -2.208639
```

```
## Predicted using Predict is:
```

```
##  [1] -1.998999  1.560584  1.000278 -1.756815 -2.669577  1.291312 -1.068444
##  [8] -1.312493  1.000184 -2.208639
```

```
## Are both predictions equal?
```

```
## [1] TRUE
```

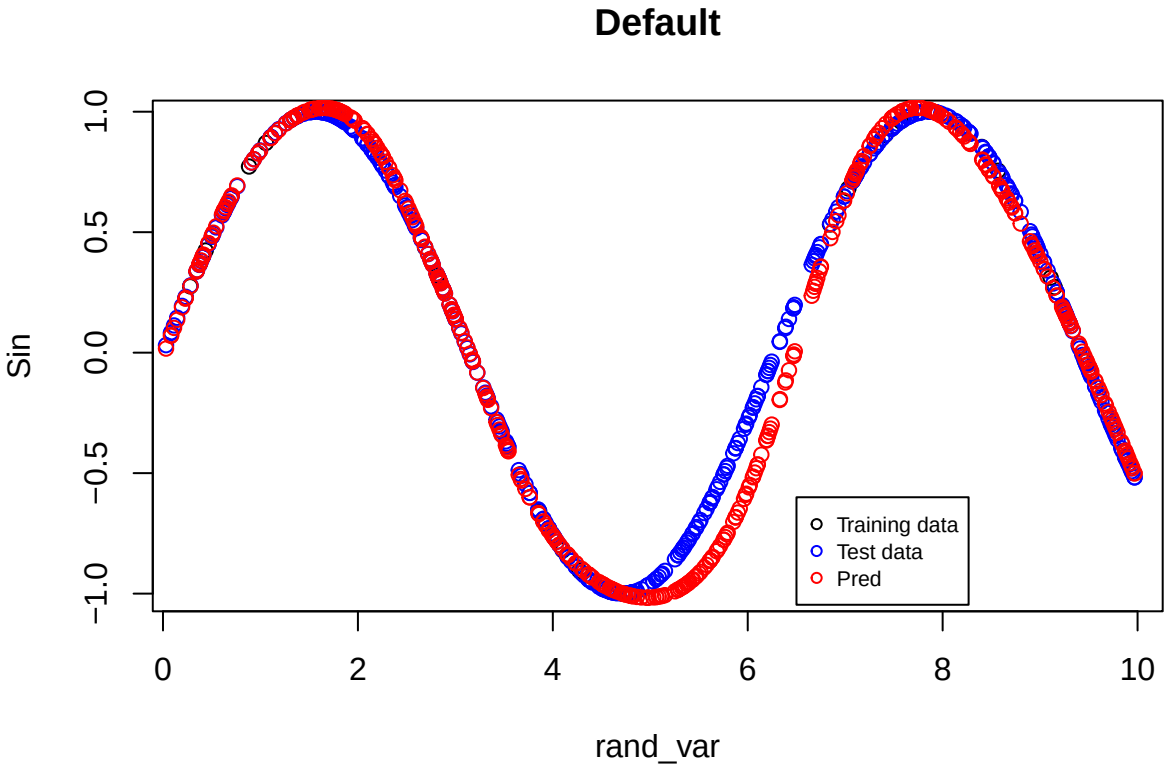
Assignment 3: NEURAL NETWORKS(Solved by Daniele Bozzoli)

Answer:

3.1

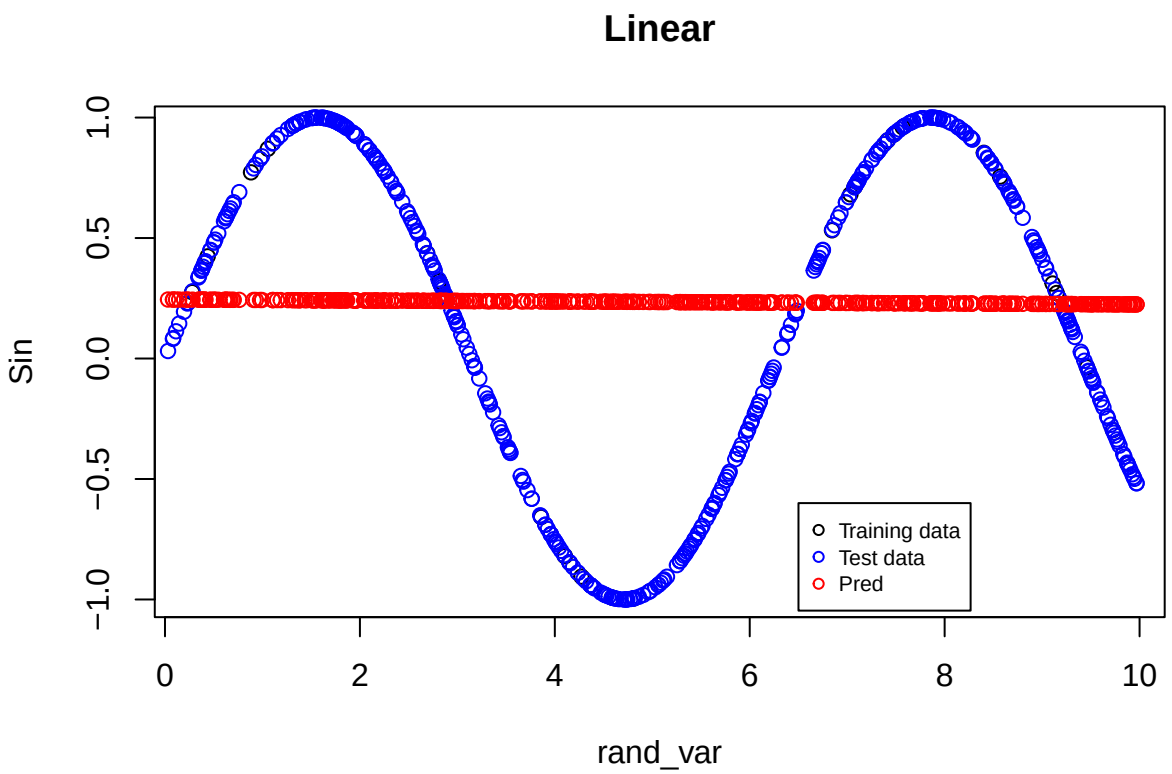
We observe in the plot that the predictions using neural networks seem to be good predictions for the sine function.

Since this function is differentiable, it also provides a smooth gradient.



3.2

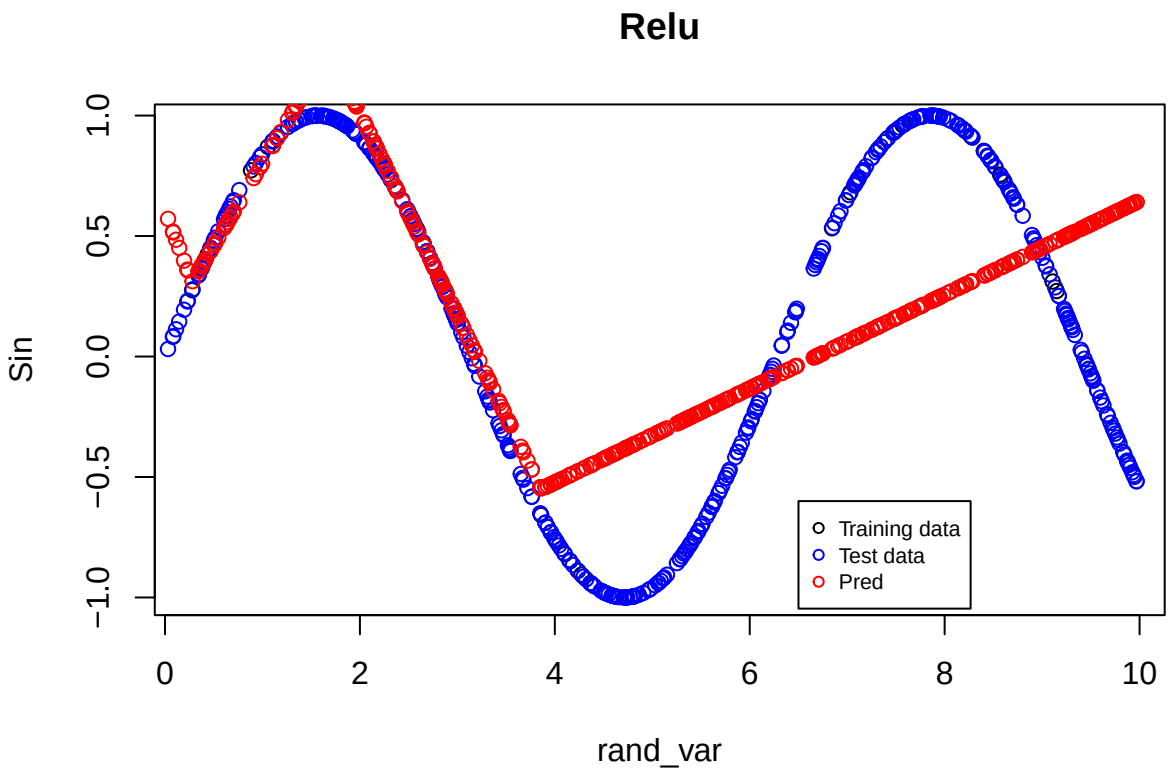
We define the linear activation function $f(x) = x$. According to the plot, we can see that the prediction is a straight line compared to the train and test data which shows a sine wave.



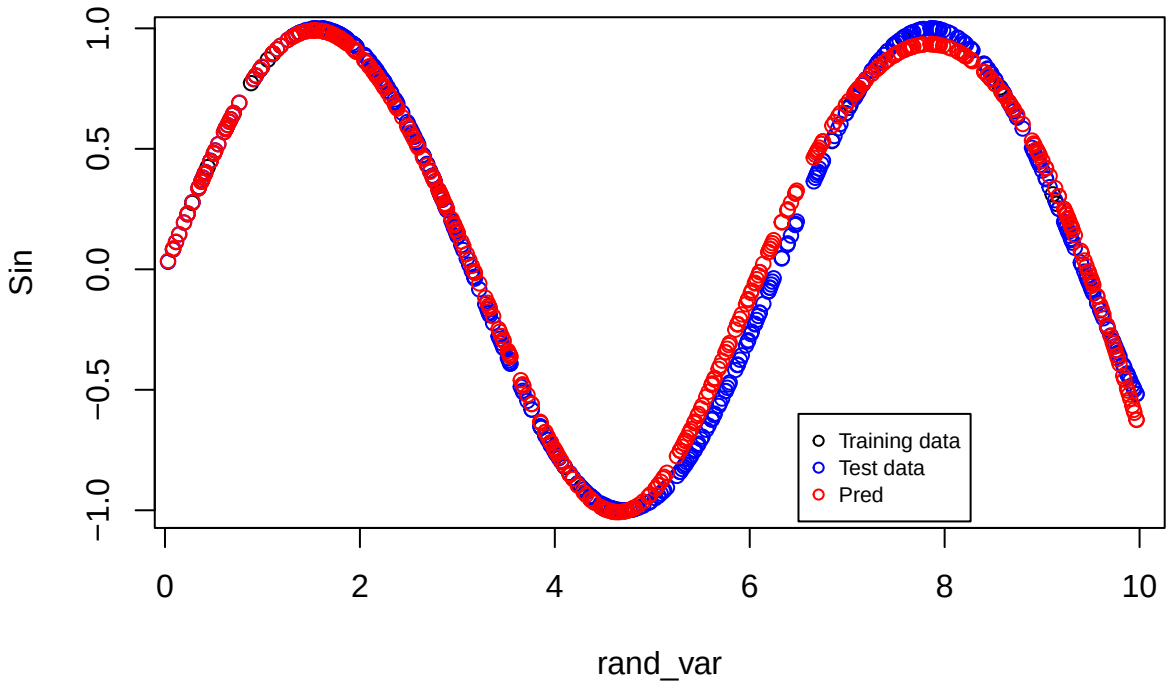
Then we define ReLU activation function $f(x) = ifelse(x >= 0, x, 0)$. We use ifelse to define the function since max(0,x) not working here.

As can be seen in the plot, ReLU function as an activation function does not provide good results on test data.

The right prediction range is only around(0.5,4). when a random variable is in range (4,10),
it shows a straight line as what we see in the linear activation function.

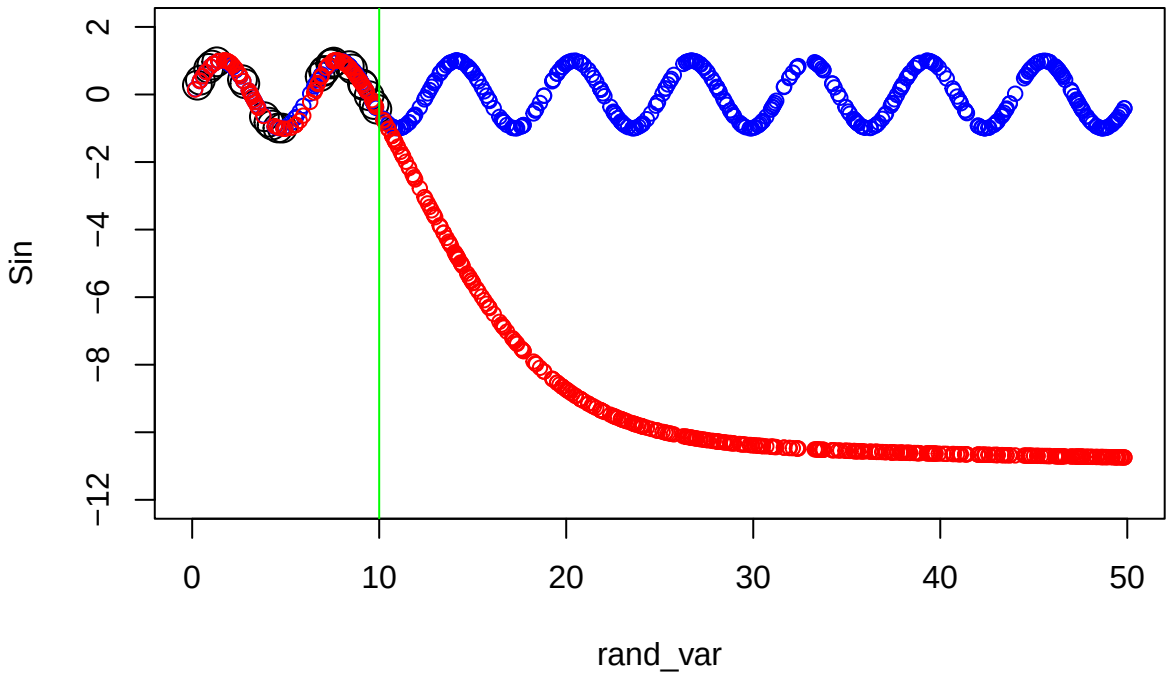


Last we define Softplus activation function $f(x) = log(1 + exp(x))$, as can be seen in the plot, the Softplus function as an activation function provides good results on test data.



3.3

The code is as follows. When random in range (0,10), the prediction matches the test data, when it is greater than 10, the prediction begins to get smaller and eventually converges to a value that is around -11.



3.4

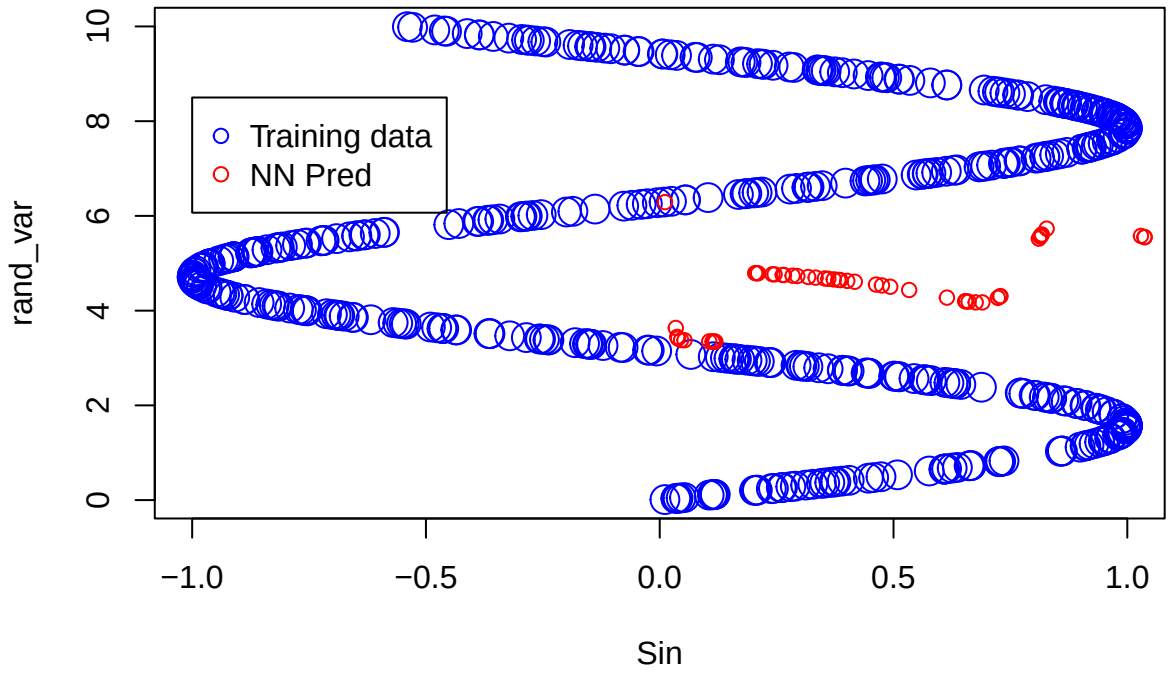
We define the sigmoid as $f(x) = (1/(1 + exp(-x)))$. by setting a relatively big value of 100, we can calculate the approximation value of the convergence to -11.00524.

It's because the original neural network is training on the sample data in the range of (0,10), when the test data is smaller than 10, the prediction looks good, but when the test data is greater than 10, the prediction will not match the test data. To Solve this problem, we can train the neural network on a bigger range of data.

```
## The value will converge to: -11.00524
```

3.5

We can see that the prediction is not good, the reason is many input values can be mapped to the same value if we use the sine function, but if we inverse this function, the original input value can not be calculated through the inverse of the sine function. so the neural network can not learn the mapping relationship between input and output and generates a bad prediction.


Training data
NN Pred
rand_var
Sin
0
2
4
6
8
10
−1.0
−0.5
0.0
0.5
1.0

Appendix: All code for this report

```
############################  Init code For Assignment 1 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(geosphere)
set.seed(1234567890)
############################ init data ######################################
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations, temps, by = "station_number")
# define the date of interest
date_interest <- as.Date("2010-1-1")

# Point to predict(Linköping University Valla Campus Geo Location)
a <- 58.3986
b <- 15.5780

# other parameters
# distance unit is KM
h_distance <- 100
h_date <- 1000
h_time <- 2
# It will generate a sequence of time from 4:00:00 to 24:00:00 with a step of 2 hours
times_interest <- c(paste0("0",seq(4,8,by=2),":00:00"), paste0(seq(10,24,by=2),":00:00"))
########################## Kernel Code  and function call ##############################

# function to predict the temperature of a geo location in Sweden given data
kernel_prediction <- function(h_distance, h_date, h_time, a, b, date, times){

  distance_vector <- c()
  distance_date_vector <- c()
  distance_time_vector <- c()

  kernel_distance_vector <- c()
  kernel_date_distance_vector <- c()
  kernel_time_distance_vector <- c()

  pred_temp_sum  <- c()
  pred_temp_mul <- c()

  for(i in 1:length(times)){

    # Filter out time that is later than data of interest
    filtered_st <- st[(st$date == date & st$time <= times[i]) | st$date < date,]

    # calaulate the distance between the point of interest and the weather station on filtered data
    # Convert to KM
    distances <- distHaversine(c(a, b), filtered_st[,c('latitude','longitude')]) / 1000
    distance_vector <- c(distance_vector,distances)
    kernel_distance <- exp(-(distances)^2 / (2 * h_distance^2))
    kernel_distance_vector <- c(kernel_distance_vector,kernel_distance)

    # calaulate the distance between the date of interest and the date on filtered data
```

```
    date_diff <- abs(as.numeric(difftime(date, filtered_st$date, units = "days")))
    distance_date_vector = c(distance_date_vector,date_diff)
    kernel_date_distance <- exp(-(date_diff)^2 / (2 * h_date^2))
    kernel_date_distance_vector <- c(kernel_date_distance_vector,kernel_date_distance)

    # calculate the distance between the time of interest and the time on filtered data
    time_diff <- abs(as.numeric(difftime(
                                strptime(filtered_st$time, "%H:%M:%S"),
                                strptime(times[i], "%H:%M:%S")),
                                units="hours"))
    distance_time_vector = c(distance_time_vector,time_diff)
    kernel_time_distance <- exp(-(time_diff)^2 / (2 * h_time^2))
    kernel_time_distance_vector <- c(kernel_time_distance_vector,kernel_time_distance)

    # calculate the sum of 3 kernels
    kernel_sum <- kernel_distance + kernel_date_distance + kernel_time_distance

    # calculate the mul of 3 kernels
    kernel_mul <- kernel_distance * kernel_date_distance * kernel_time_distance

    # predict the temperature
    predicted_temp_sum <- sum((kernel_sum * filtered_st$air_temperature) / sum(kernel_sum))
    predicted_temp_mul <- sum((kernel_mul * filtered_st$air_temperature) / sum(kernel_mul))

    pred_temp_sum <- c(pred_temp_sum,predicted_temp_sum)
    pred_temp_mul <- c(pred_temp_mul,predicted_temp_mul)
  }
  return(list(pred_temp_sum,pred_temp_mul,
              distance_vector,kernel_distance_vector,
              distance_date_vector,kernel_date_distance_vector,
              distance_time_vector,kernel_time_distance_vector))
}

kernel_result <- kernel_prediction(h_distance, h_date, h_time, a, b, date_interest, times_interest)

################################## 1.3 plot ##########################################
# plot the distance vs kernel_distance
plot(kernel_result[[3]],kernel_result[[4]],xlab="geo distance",
     ylab="kernel_geo_distance",main="Geo distance vs kernel_distance")

# plot the date distance vs kernel_distance
plot(kernel_result[[5]],kernel_result[[6]],xlab="date distance",
     ylab="kernel_date_distance",main="date distance vs kernel_date_distance")

# plot the time distance vs kernel_distance
plot(kernel_result[[7]],kernel_result[[8]],xlab="time distance",
     ylab="kernel_time_distance",main="time distance vs kernel_time_distance")
# predict the temperature using sum kernel
pred_temp_sum <- kernel_result[[1]]

# predict the temperature using mul kernel
pred_temp_mul <- kernel_result[[2]]
```

```r
# print out the predicted temperature of the sum kernel
cat("The predicted temperature of the sum kernel:\n")
pred_temp_sum

# Print out the predicted temperature of mul kernel
cat("The predicted temperature of the mul kernel:\n")
pred_temp_mul
plot(seq(4,24,by=2),pred_temp_sum,type="o",xlab="time",ylim=c(-5,10),col="red",
     ylab="temperature",main="time vs temperature prediction")
lines(seq(4,24,by=2),pred_temp_mul,type="o",col="blue")
legend(x=13,y=0,
       legend = c("sum kernel","mul kernel"),
       col=c("red", "blue"),
       lty=c(1,1))
############################  Init code For Assignment 2 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(kernlab)
set.seed(1234567890)
############################  Load Data and split data sets ####################
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, ]

############################  Get error va and corresponding C ##################
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))
}

c1 <- which.min(err_va)

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)

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)
```

```
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)


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)

cat("error rate of filter0 is", (err0)*100,"%\n")
cat("error rate of filter1 is", (err1)*100,"%\n")
cat("error rate of filter2 is", (err2)*100,"%\n")
cat("error rate of filter3 is", (err3)*100,"%\n")
err_va <- NULL
for(i in seq(by,5,by)){
  filter <- ksvm(type~.,data=trva,kernel="rbfdot",kpar=list(sigma=0.05),C=i,scaled=FALSE)
  mailtype <- predict(filter,te[,-58])
  t <- table(mailtype,te[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}

c2 <- which.min(err_va)

cat("C based on Tr + Va is ", c1*by,"\n")
cat("C based on TrVa + Te is", c2*by,"\n")
filter0_pred <- predict(filter0, te, type = "response")
filter0_gerr <- mean(filter0_pred != te[, 58])

filter1_pred <- predict(filter1, te, type = "response")
filter1_gerr <- mean(filter1_pred != te[, 58])

filter2_pred <- predict(filter2, te, type = "response")
filter2_gerr <- mean(filter2_pred != te[, 58])

filter3_pred <- predict(filter3, te, type = "response")
filter3_gerr <- mean(filter3_pred != te[, 58])

cat("generalization error of filter0 is", filter0_gerr*100,"%\n")
cat("generalization error of filter1 is", filter1_gerr*100,"%\n")
cat("generalization error of filter2 is", filter2_gerr*100,"%\n")
cat("generalization error of filter3 is", filter3_gerr*100,"%\n")
sv <- alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
# RBF kernel with sigma = 0.05
rbf <- rbfdot(sigma = 0.05)

pred <- c()
```

```r
# We produce predictions for just the first 10 points in the dataset.
for(i in 1:10){
  k <- c()
  for(j in 1:length(sv)){
    x <- as.vector(unlist(spam[i, -58])) # test data
    y <- as.vector(unlist(spam[sv[j], -58])) # support vector
    k <- c(k, rbf(x, y))
  }
  pred<-c(pred, (k %*% co) + inte)
}

pred_using_func <- as.vector(predict(filter3, spam[1:10,-58], type = "decision"))

cat("Calculated Prediction is:")
pred

cat("Predicted using Predict is:")
pred_using_func

cat("Are both predictions equal?")
all.equal(pred, pred_using_func)
############################  Init code For Assignment 3 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(neuralnet)
set.seed(1234567890)
##########################  3.1 ################################################
rand_var <- runif(500, 0, 10)
df <- data.frame(rand_var, Sin=sin(rand_var))
train <- df[1:25,] # Training
test <- df[26:500,] # Test

# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
nn <- neuralnet(data = train, formula = Sin ~ rand_var, hidden = c(10), startweights = winit)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1, main="Default")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)
##########################  3.2.1 Linear ######################################

#Linear
h1 <- function(x) {
  x
}

nn1 <- neuralnet(data = train, formula = Sin ~ rand_var, hidden = c(10), act.fct = h1)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1, main="Linear")
```

```
points(test, col = "blue", cex=1)
points(test[,1], predict(nn1,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)

############################  3.2.2 Relu ########################################
#ReLU
h2 <- function(x) {
  # max(0,x) not working
  ifelse(x>=0,x,0)
}

nn2 <- neuralnet(formula = Sin ~ rand_var, data = train, hidden = 10,
startweights = winit, act.fct = h2)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1,main="Relu")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn2,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)
############################  3.2.3 Softplus ########################################

#Softplus
h3 <- function(x) {
  log(1+exp(x))
}

nn3 <- neuralnet(formula = Sin ~ rand_var, data = train, hidden = 10,
startweights = winit, act.fct = h3)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1,main="Softplus")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn3,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)

############################  3.3  ########################################
# Sample 500 points
set.seed(1234567890)
rand_var <- runif(500, min = 0, max = 50)
df <- data.frame(rand_var, Sin = sin(rand_var))

plot(train, cex = 2, xlim = c(0,50), ylim = c(-12, 2))
points(df, col = "blue", cex = 1)
points(df[, 1], predict(nn, df), col = "red", cex = 1)
abline(v = 10, col = "green")
############################  3.4  ########################################

sigmoid <- function(x){
  return(1 / (1 + exp(-x)))
}
```

```r
#
weight_1 <- nn$weights[[1]][[1]][2,]
bias_1 <- nn$weights[[1]][[1]][1,]

sigmoid_val <- sigmoid(weight_1 * 100 + bias_1)

bias_2 <- nn$weights[[1]][[2]][1,]
weight_2 <- nn$weights[[1]][[2]][2:11,]
cat("The value will converge to:", weight_2 %*% sigmoid_val + bias_2, "\n")
##########################  3.5  ######################################
# Sample 500 points
rand_var <- runif(500, min = 0, max = 10)
df <- data.frame(Sin = sin(rand_var), rand_var)
nn4 <- neuralnet(formula = rand_var ~ Sin,
                 data = df, hidden = 10, startweights = winit, threshold = 0.1)
pred_4 <- predict(nn4, df)

plot(df, cex = 2, col = "blue")
points(df[, 2], pred_4, col = "red", cex = 1)
legend(x = -1, y = 8.5, legend = c("Training data", "NN Pred"),
col = c("blue", "red"), cex = 1, pch = 1)
```