BLOCK1_LAB3

Jin Yan; Dongwei Ni; Collins Klinsmann Osondu

2022-12-16

contribution statement:

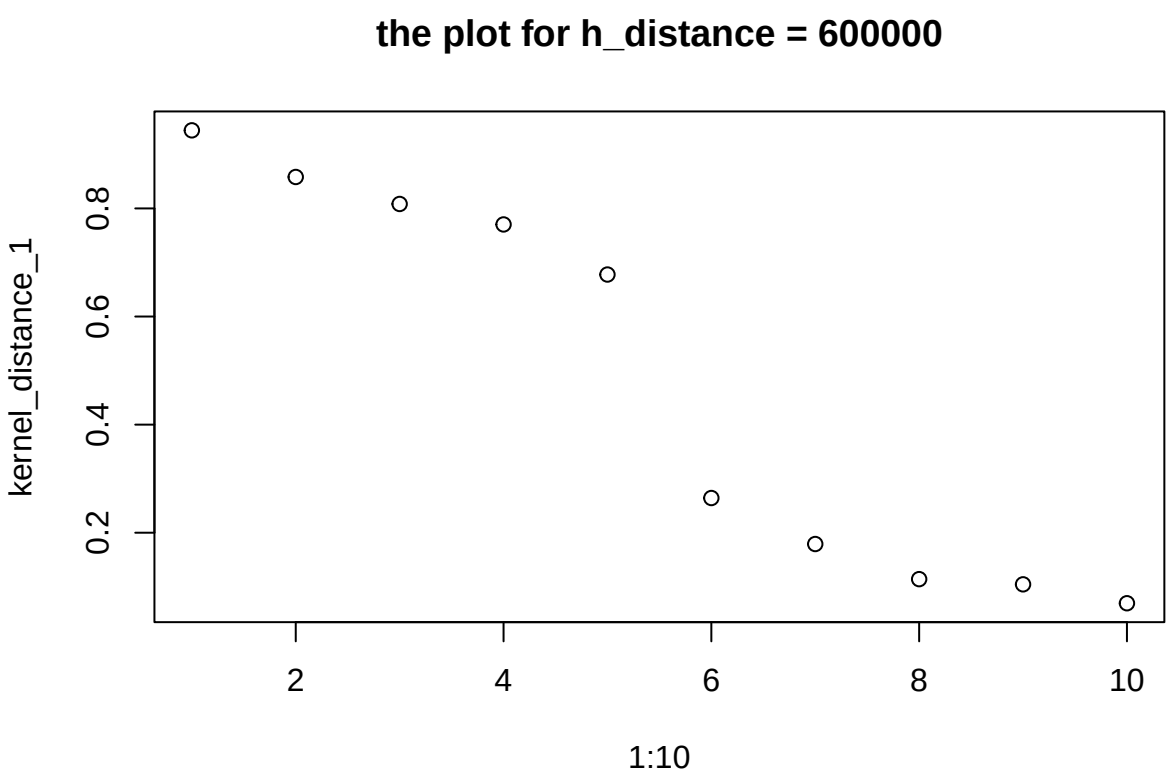
Jin Yan was responsible for the first two parts of this assignment.

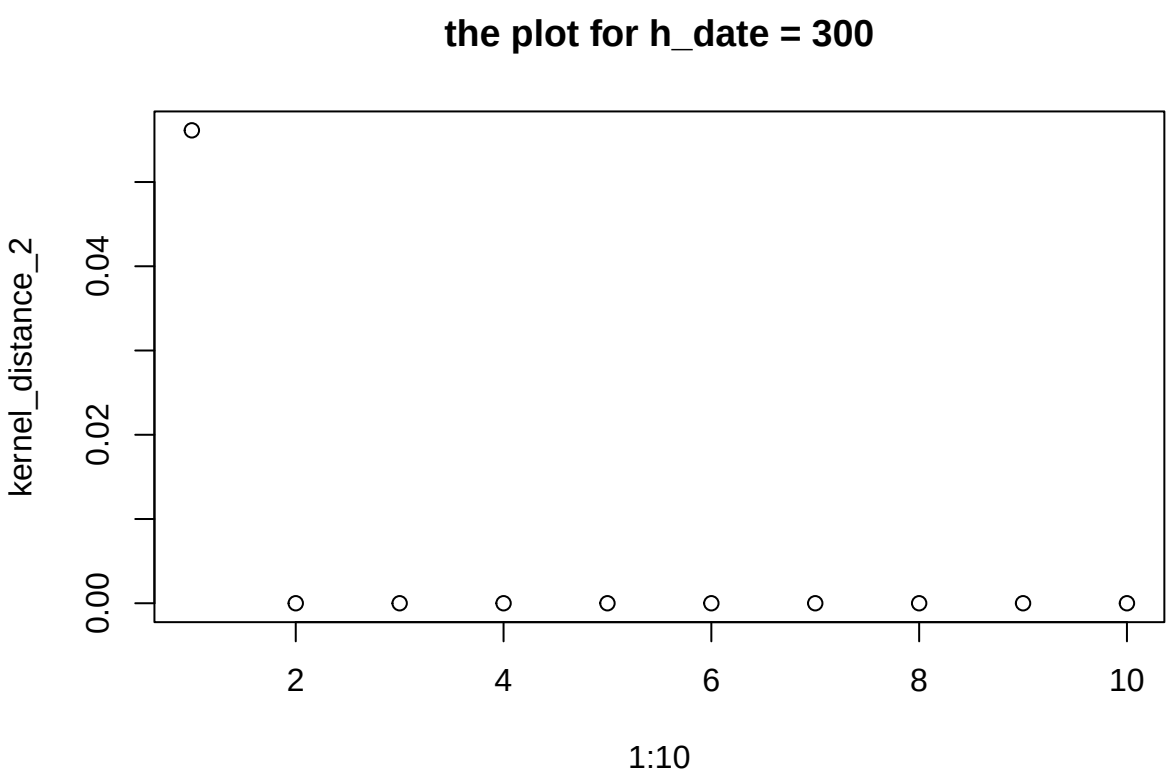
Dongwei Ni finished the third part of the assignment.

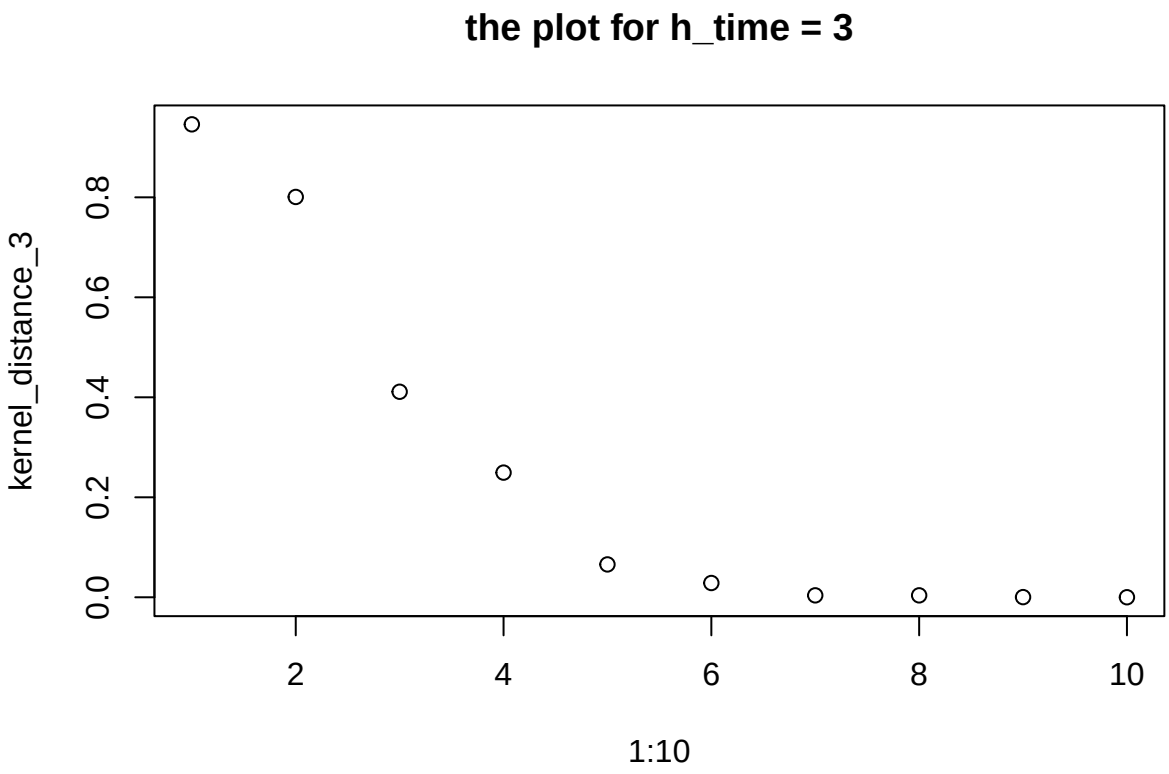
QUESTION ONE:1. KERNEL METHODS

Choose an appropriate smoothing coefficient or width for each of the three kernels above.No cross-validation should be used. Instead, choose manually a width that gives large kernel values to closer points and small values to distant points. Show this with a plot of the kernel value as a function of distance

In this task I choose h_distance = 600000,h_date = 300,h_time = 3. The following plots can prove these distances can satisfy the requirements giving in the question(gives large kernel values to closer points and small values to distant points).

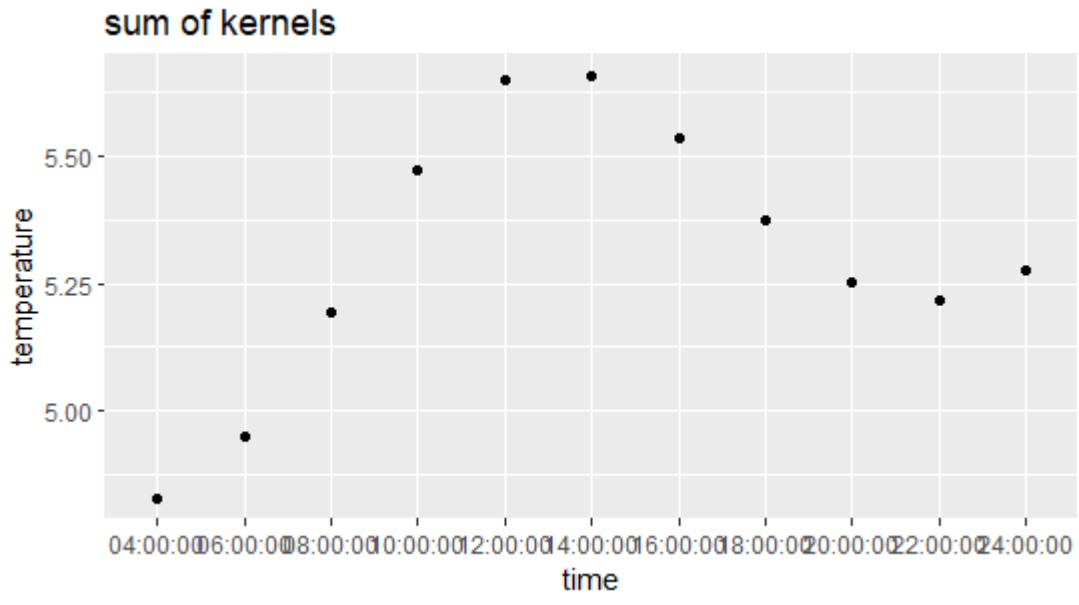

the plot for h_distance = 600000
kernel_distance_1
0.2
0.4
0.6
0.8
2
4
6
8
10
1:10


the plot for h_date = 300
kernel_distance_2
0.00
0.02
0.04
2
4
6
8
10
1:10



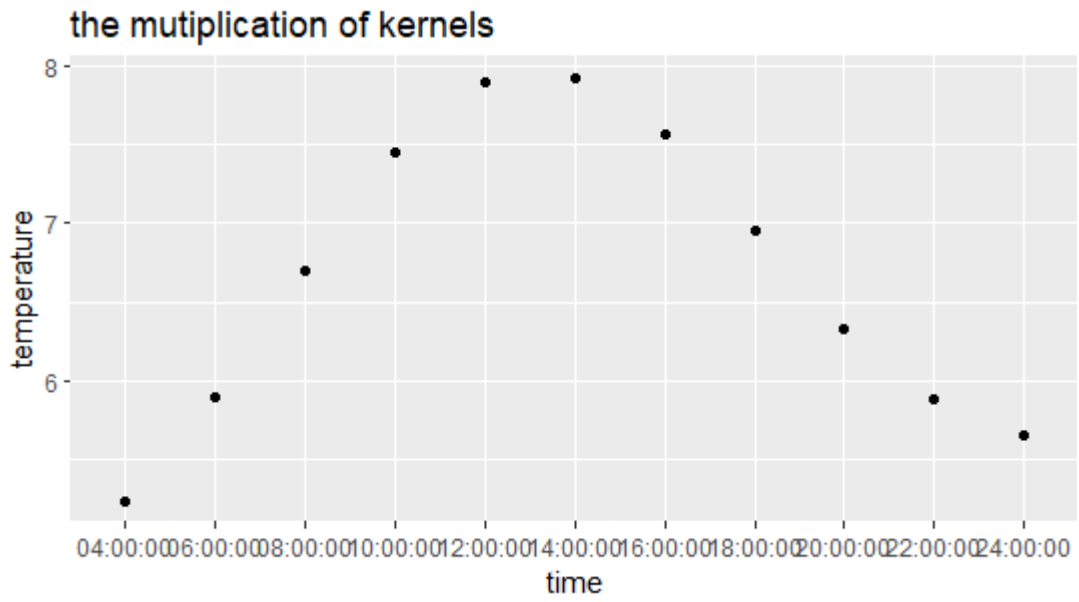
the comparision between two ways of combining different kernels

combine kernels by summing them up



*

combine kernels by multipying them



*

From the two plots we can clearly see that the second prediction is better. I think the main reason is relevant to this specific task. In this task, we use different parts of input to create three kernels with one type of kernel instead of using all the parts of the input with more than one type kernel.

2. SUPPORT VECTOR MACHINES

1. Which filter do we return to the user ? filter0, filter1, filter2 or filter3? Why? After calculation, we find that

err0 = 0.0675 err1 = 0.08489388 err2 = 0.082397 err3 = 0.02122347

The filter1 should be returned to the user. The main reason is the error is the lowest, which means it has high precision, for making prediction.

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

I think err1 should be returned to the user. This is because the generalization error refers to the error we get when we use the mode to make prediction according to new unseen data set. Only err1 is created in this way. Namely, we use training data set to train the model and then use test data set to make prediction.

3. Implementation of SVM predictions.

After calculation we can find through different methods, we find the same prediction. The blue dots in the graph are created with a linear combination. The red line is created with “predict” function.

*

To calculate the result of linear combination, we used the following formula.

$$\widehat{y}(\mathbf{x}_\star) = \text{sign}\left(\widehat{\boldsymbol{\alpha}}^\top \boldsymbol{K}(\mathbf{X}, \mathbf{x}_\star)\right)$$

*

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\ell^2}\right),$$

*

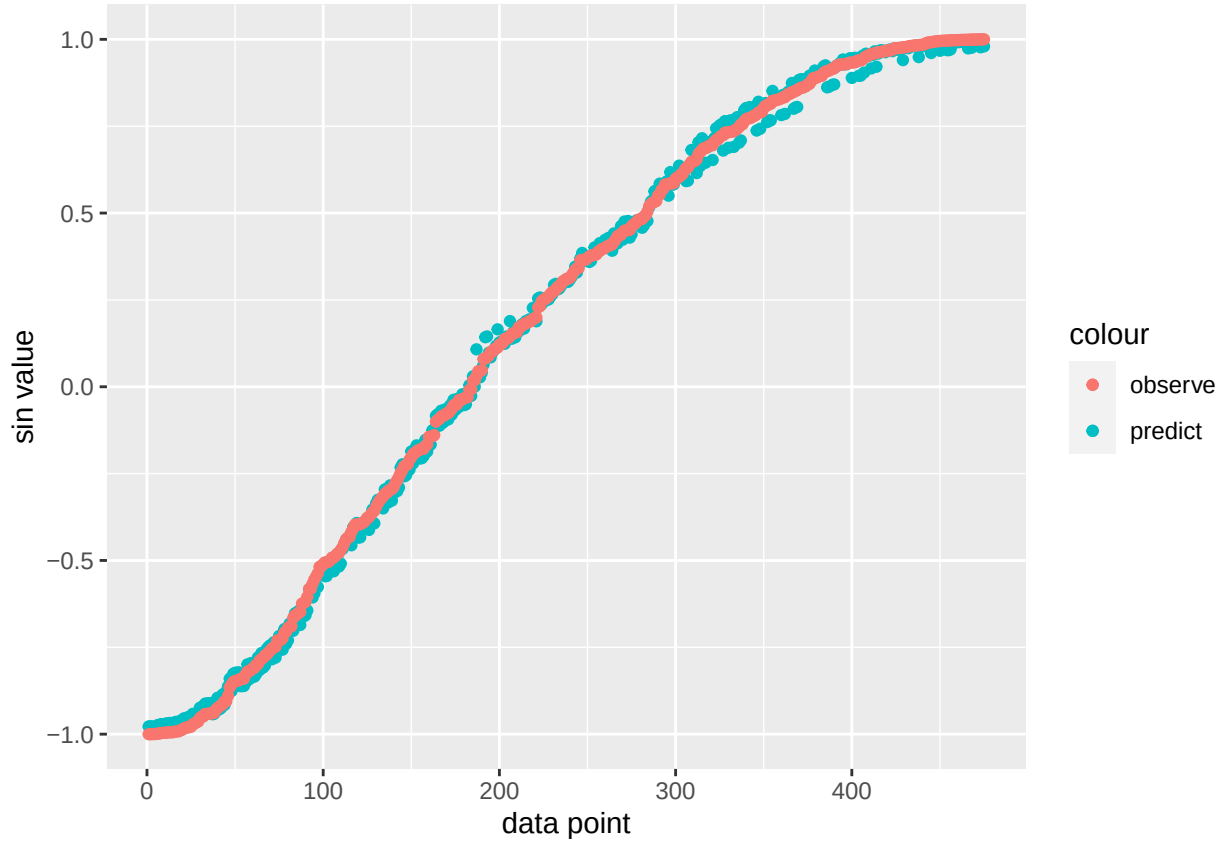
$$\boldsymbol{K}(\mathbf{X}, \mathbf{x}_\star) = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_\star) \\ \kappa(\mathbf{x}_2, \mathbf{x}_\star) \\ \vdots \\ \kappa(\mathbf{x}_n, \mathbf{x}_\star) \end{bmatrix}.$$

*

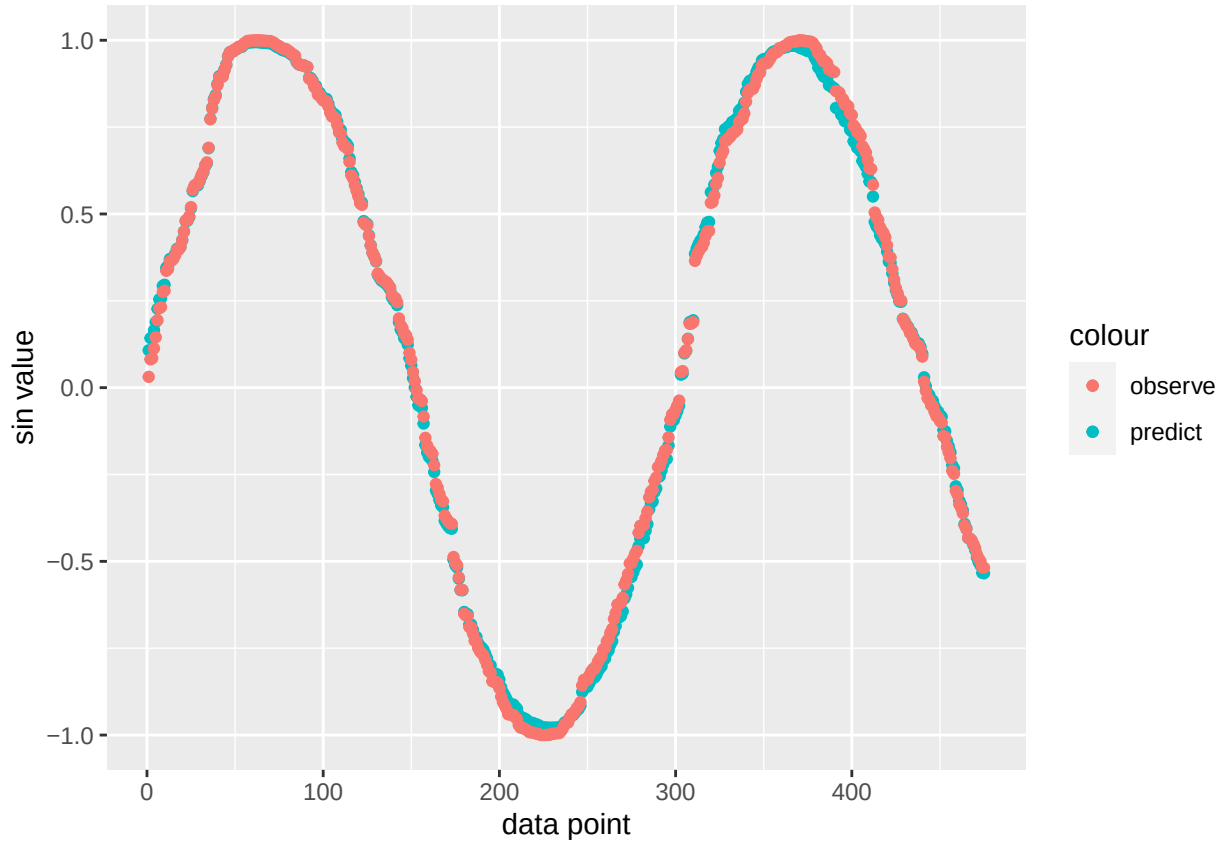
3.NEURAL NETWORKS

Task 1

```
set.seed(1234567890)

data1 <- runif(500,0,10)
data1sin <- sin(data1)

dataSet <- tibble(data1, data1sin)

sampleTrain <- sample(1:500,25)
train <- dataSet[sampleTrain,]
test <- dataSet[-sampleTrain,]

NNtrain <- neuralnet(data1sin~data1, data = train,hidden = 10)

Fit_dataSet <- predict(NNtrain,newdata = test)

dfggplot <- data.frame(Fit_dataSet, test$data1sin) %>%
  arrange(test.data1sin)
ggplot(data = dfggplot)+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot[,1], color = "predict"))+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot[,2], color = "observe"))+
  labs(x = "data point", y = "sin value")
```



```
dfggplot2 <- data.frame(Fit_dataSet, test) %>%
  arrange(data1)
ggplot(data = dfggplot2)+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot2[,1], color = "predict"))+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot2[,3], color = "observe"))+
  labs(x = "data point", y = "sin value")
```

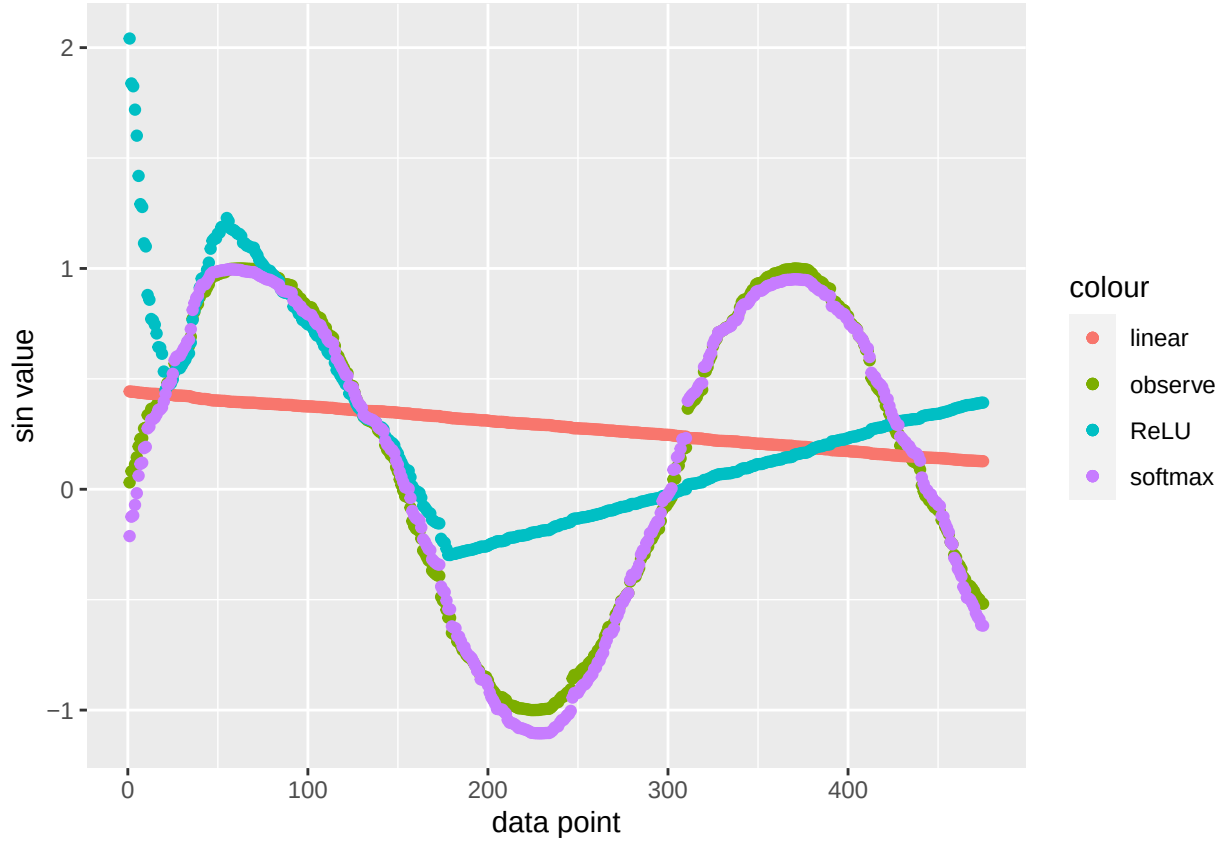


- As we can see from the plot ,the NN model works great. Even though there are some subtle phase-shift-like differences when sin value is close to 1 or -1, its in general a good model.

Task 2

```
h1_linear <- \(x) x
h2_ReLU <- \(x) ifelse(x >= 0, x, 0)
h3_softmax <- \(x) log(1+exp(x))

NNh1 <- neuralnet(data1sin~data1, data = train, hidden = 10, act.fct = h1_linear)
NNh2 <- neuralnet(data1sin~data1, data = train, hidden = 10, act.fct = h2_ReLU,
                  learningrate.limit = c(0,0.01))
# Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
NNh3 <- neuralnet(data1sin~data1, data = train, hidden = 10, act.fct = h3_softmax,
                  learningrate.limit = c(0,0.01))
```

```
fit_NNh1 <- predict(NNh1, newdata = test)
fit_NNh2 <- predict(NNh2, newdata = test)
fit_NNh3 <- predict(NNh3, newdata = test)

Q2dfggplot <- data.frame(test, fit_NNh1, fit_NNh2, fit_NNh3)
Q2dfggplot <- Q2dfggplot %>% arrange(data1)
ggplot(data = Q2dfggplot)+
  geom_point(aes(x = 1:length(fit_NNh1), y = data1sin, color = "observe"))+
  geom_point(aes(x = 1:length(fit_NNh1), y = fit_NNh1, color = "linear"))+
  geom_point(aes(x = 1:length(fit_NNh1), y = fit_NNh2, color = "ReLU"))+
  geom_point(aes(x = 1:length(fit_NNh1), y = fit_NNh3, color = "softmax"))+
  labs(x = "data point", y = "sin value")
```

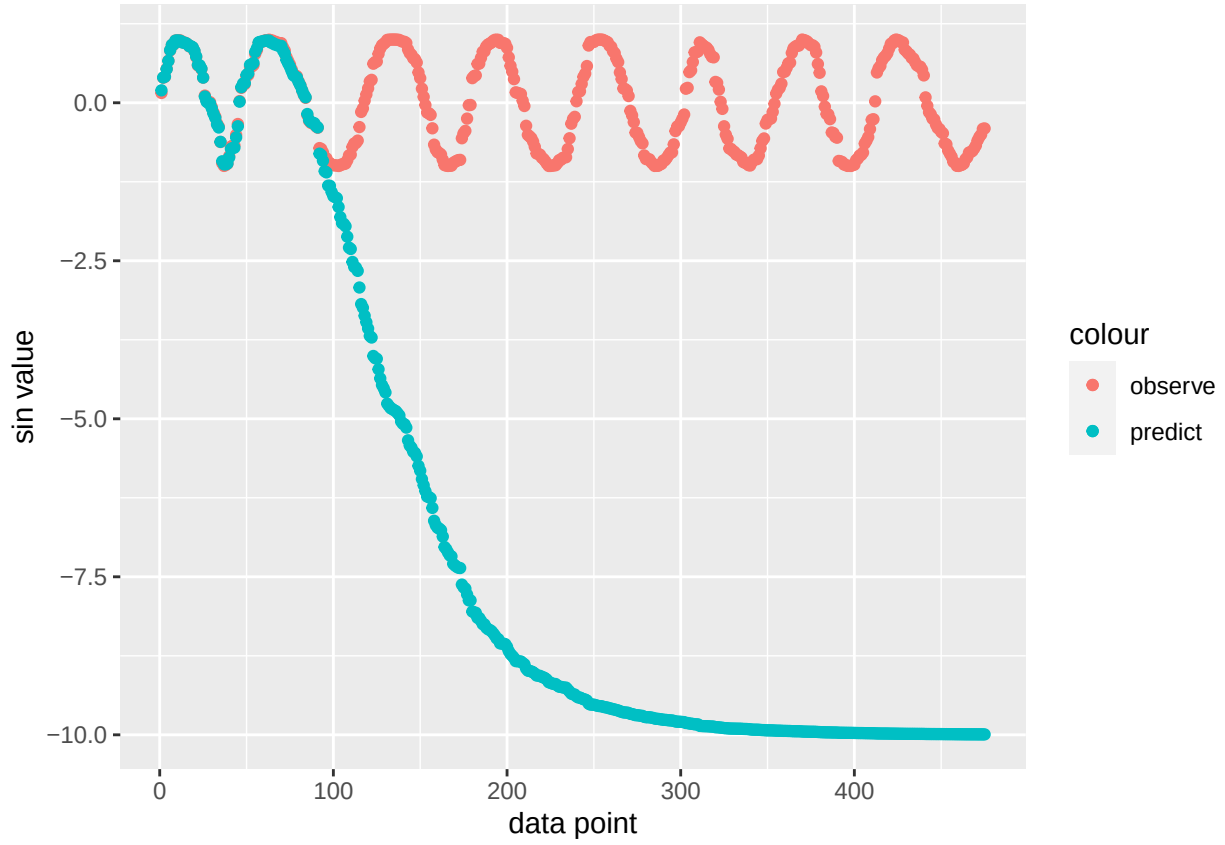


- From the plot, we can find that for our model, linear and ReLU doesn't work well as activation functions, only softmax did a proper job

Task 3

```
set.seed(1234567890)

data2 <- runif(500,0,50)
data2sin <- sin(data2)
```

```
dataSet2 <- tibble(data1 = data2, data1sin = data2sin)

sampleTrain <- sample(1:500,25)
train2 <- dataSet2[sampleTrain,]
test2 <- dataSet2[-sampleTrain,]

fit_Q3te <- predict(NNtrain, newdata = test2)

dfggplotQ3 <- data.frame(test2, fit_Q3te) %>% arrange(data1)

ggplot(data = dfggplotQ3)+
  geom_point(aes(x = 1:length(fit_Q3te), y = data1sin, color = "observe"))+
  geom_point(aes(x = 1:length(fit_Q3te), y = fit_Q3te, color = "predict"))+
  labs(x = "data point", y = "sin value")
```



- It only works within [1,10] and cannot handle those point sampled from (10,50] at all. The predicted value just follow with the trend of which when training set ends(i.e. when approaching from the left to x = 10). And converge to sin(x) = -10.

Task 4

```
NNtrain$weights
```

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

```
sum(NNtrain$weights[[1]][[2]][which(NNtrain$weights[[1]][[1]][2,] >0) + 1,1]) + NNtrain$weights[[1]][[2
```

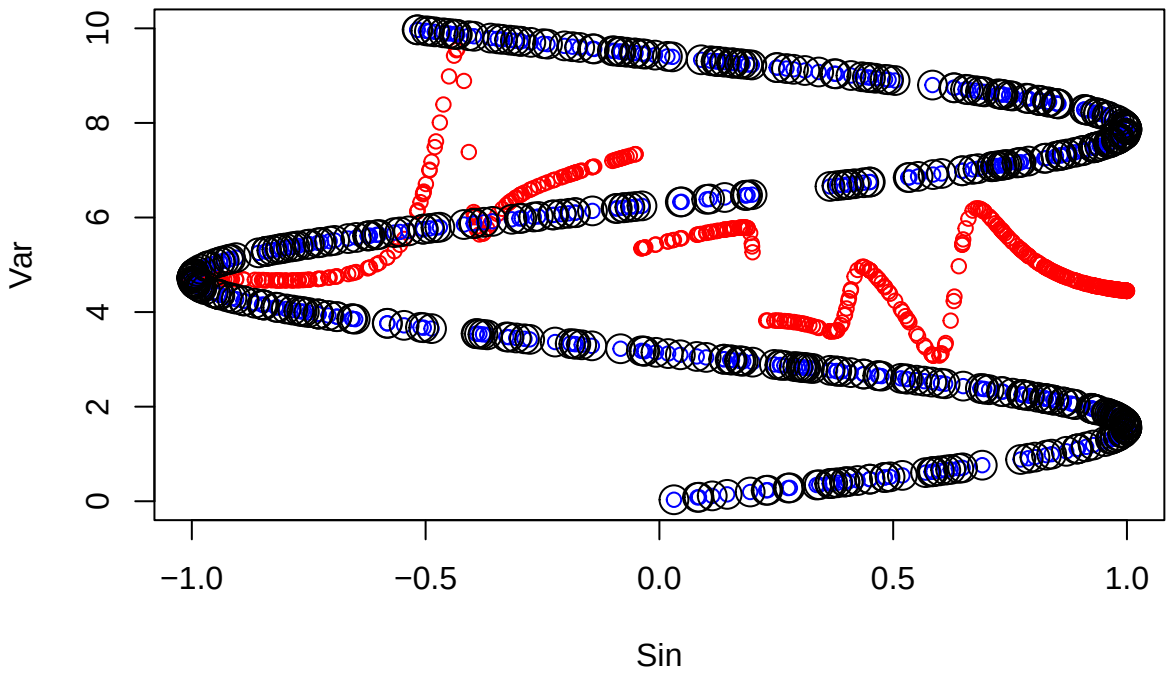
```
## [1] -10.00631
```

- We are using sigmoid as activation function(by default). When x grows greater, sigmoid will return either a number that close to 1(under positive weight) or 0 (under negative weight). Thus for the corresponding output weights, we can figure out that the convergence is towards -10.00631.

Task 5

```
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata
te <- mydata

NNQ5 <- neuralnet(Var~Sin, mydata, hidden = 10, threshold = 0.1)
```

```
plot(te[,2],predict(NNQ5,te), col="red", cex=1, type = "p",
     ylim = c(0,10), xlab = "Sin", ylab = "Var")
points(te[,2],te[,1], col = "blue", cex=1)
points(tr[,2], tr[,1],col = "black", cex=2)
```



- When mapping x towards sin(x), one certain x only corresponds one certain sin(x), but when mapping sin(x) towards x, one sin(x) will respond multiple x values. That makes the model don't work at all.

APPENDIX

TASK1

```
set.seed(1234567890)
library(geosphere)
library(ggplot2)
stations <- read.csv("C:/Users/yj313/Desktop/LAB
                     /732A99-MachineLearning/lab03/part1/stations.csv")
temps <- read.csv("C:/Users/yj313/Desktop/LAB/
                  732A99-MachineLearning/lab03/part1/temps50k.csv")
st <- merge(stations,temps,by="station_number")

# create three functions to calculate three kinds of distances between the point
# of interest and the training point.

# distance_1 <- function(point1,point2){
#   return(distHaversine(point1,as.matrix(point2,ncol = 2)))
# }
distance_1 <- function(point1,point2){
  return(distHaversine(point1,point2))
}
```

```r
distance_2 <- function(date1,date2){
  days <- difftime(as.POSIXct(date1,format= "%Y-%m-%d"),
                   as.POSIXct(date2,format = "%Y-%m-%d"),units = "days")
  return(as.numeric(days))
}


distance_3 <- function(time1,time2){
  hours <- difftime(as.POSIXct(time1,format = "%H:%M:%S"),
                    as.POSIXct(time2,format = "%H:%M:%S"),units = "hours")
  return(as.numeric(hours))
}


# using plots to show widths chosen are appropriate

new_index <- sample(nrow(st))
new_data <- st[new_index,]
data_shown <- new_data[1:10,]

# for the first width

new_data_1 <-  c(" 58.4274", "14.826", "2013-11-04","08:00:00")
h_distance <- 600000
distance <- NULL
for(i in 1:10){
  distance_temp <- distance_1(as.numeric(new_data_1[1:2]),data_shown[i,4:5])
  distance <- c(distance,distance_temp)
}

order_ascend <- order(distance)
data_distance_1 <- data_shown[order_ascend,]

kernel_distance_1 <- NULL

for (i in 1:10) {
  kernel_distance_1_temp <- exp(-distance_1(as.numeric(new_data_1[1:2]),
                                data_distance_1[i,4:5])^2 / (2 * h_distance^2))
  kernel_distance_1 <- c(kernel_distance_1,kernel_distance_1_temp)
}

plot(1:10,kernel_distance_1,main = "the plot for h_distance = 600000")


# for second width

new_data_2 <-  c("58.4274", "14.826", "2013-11-04","08:00:00")
h_date <- 300
distance <- NULL
for(i in 1:10){
  distance_temp <- distance_2(new_data_2[3],data_shown[i,9])
  distance <- c(distance,distance_temp)
}
```

```r
order_ascend <- order(distance)
data_distance_2 <- data_shown[order_ascend,]

kernel_distance_2 <- NULL

for (i in 1:10) {
  kernel_distance_2_temp <- exp(-distance_2(new_data_2[3],
                                    data_distance_2[i,9])^2 / (2 * h_date^2))
  kernel_distance_2 <- c(kernel_distance_2,kernel_distance_2_temp)
}

plot(1:10,kernel_distance_2,main = "the plot for h_date = 300")


# for thrid width

new_data_3 <-  c("58.4274", "14.826", "2013-11-04","08:00:00")
h_time <- 3
distance <- NULL
for(i in 1:10){
  distance_temp <- distance_3(new_data_2[4],data_shown[i,10])
  distance <- c(distance,distance_temp)
}

order_ascend <- order(abs(distance))
data_distance_3 <- data_shown[order_ascend,]

kernel_distance_3 <- NULL

for (i in 1:10) {
  kernel_distance_3_temp <- exp(-distance_3(new_data_3[4],
                                    data_distance_3[i,10])^2 / (2 * h_time^2))
  kernel_distance_3 <- c(kernel_distance_3,kernel_distance_3_temp)
}

plot(1:10,kernel_distance_3,main = "the plot for h_time = 3")




predicted_temp <- function(data1,data2,h_distance = 600000,
                           h_date = 300, h_time = 3){
  n <- nrow(data2)
  nominator<- NULL
  denominator <- NULL

  nominator_2 <- NULL
  denominator_2 <- NULL

  for(i in 1:n){
    kernel_1 <- exp(-distance_1(as.numeric(data1[1:2]),
                                data2[i,4:5])^2 / (2 * h_distance^2))
```

```
    kernel_2 <- exp(-distance_2(data1[3],data2[i,9])^2 / (2 * h_date^2))
    kernel_3 <- exp(-distance_3(data1[4],data2[i,10])^2 / (2 * h_time^2))

    kernel_real <- kernel_1 * kernel_2 * kernel_3
    nominator_temp <- kernel_real * data2[i,11]
    denominator_temp <- kernel_real
    nominator <- c(nominator,nominator_temp)
    denominator <- c(denominator,denominator_temp)

    kernel_real_2 <- kernel_1 +  kernel_2 + kernel_3
    nominator_temp_2 <- kernel_real_2 * data2[i,11]
    denominator_temp_2 <- kernel_real_2
    nominator_2 <- c(nominator_2,nominator_temp_2)
    denominator_2 <- c(denominator_2,denominator_temp_2)
  }
  y_hat <- sum(nominator) /sum(denominator)
  y_hat_2 <- sum(nominator_2) /sum(denominator_2)
  return(c(y_hat,y_hat_2))

}

latitude <- rep("58.4274",11)
longitude <- rep("14.826",11)
date <- rep("2013-11-04",11)
day_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")
table <- data.frame(latitude,longitude,date,day_times)

temperature_one <- NULL
temperature_two <- NULL

for(i in 1:11){
  data_interest <- as.vector(table[i,])
  # the training data consist of two parts, one part is data created before
  # the date we want to predict for. Another part is that data we collected
  # during that day but not at the same time as that
  # we want to predict for.

  data_needed_part1 <- st[which(st$date < "2013-11-04"),]
  data_needed_part2 <- st[which(st$date == "2013-11-04" &
                                  distance_3(st$time,data_interest[[4]]) < 0),]
  data_needed <- rbind(data_needed_part1, data_needed_part2)

  temperature_combination <-predicted_temp(c(data_interest[[1]],
          data_interest[[2]],data_interest[[3]],data_interest[[4]]),data_needed)

  temperature_one <-c(temperature_one,temperature_combination[2])
  temperature_two <-c(temperature_two,temperature_combination[1])
}


data_plot1 <- data.frame(day_times, temperature_one)
data_plot2 <- data.frame(day_times, temperature_two)
```

```
plot1 <- ggplot(data_plot1, aes(x = day_times, y = temperature_one)) +
  geom_point() +
  xlab("time") + ylab("temperature") +labs(title = "sum of kernels")
plot2 <- ggplot(data_plot2,aes(x = day_times, y = temperature_two)) +
  geom_point() +
 xlab("time") + ylab("temperature")+labs(title = "the mutiplication of kernels")

plot1
plot2
```

TASK2

```
# Lab 3 block 1 of 732A99/TDDE01/732A68 Machine Learning
# Author: jose.m.pena@liu.se
# Made for teaching purposes

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))
}
# err_va
# which.min(err_va)
# seq(by,5,by)

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

# same training dataset but this is used for predicting test data set
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

# the training data set is changed into trva
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

# the training date set is changed into spam
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)

support_vectors <- spam[sv,-58]
num_vector <- nrow(support_vectors)
y_hat <- c()
for(i in 1:10){
  kernel <- c()
  for (j in 1:num_vector) {
    kernel_temp <- exp(sum((support_vectors[j,] - spam[i,-58])**2) * -0.05)
    kernel <- c(kernel,kernel_temp)
  }
  y_hat_temp <- sum(t(kernel) * co) + inte
  y_hat <- c(y_hat,y_hat_temp)
}

y_hat2 <- predict(filter3,spam[1:10,-58], type = "decision")
data <- data.frame(x = 1:10, y = y_hat)
data2 <- data.frame(x = 1:10, y = y_hat2)
ggplot(data = data, aes( x = x, y = y)) + geom_point(color = "blue", size = 3) +
  geom_line(
  data = data2, color = "red", size = 1) +
  xlab("the index of the test data point") +
  ylab("the predicted value")
```

TASK3

```
library(neuralnet)
library(dplyr)
library(ggplot2)

library(sigmoid)
```

```r
library(NeuralNetTools)
set.seed(1234567890)

data1 <- runif(500,0,10)
data1sin <- sin(data1)

dataSet <- tibble(data1, data1sin)

sampleTrain <- sample(1:500,25)
train <- dataSet[sampleTrain,]
test <- dataSet[-sampleTrain,]

NNtrain <- neuralnet(data1sin~data1, data = train,hidden = 10)

Fit_dataSet <- predict(NNtrain,newdata = test)

dfggplot <- data.frame(Fit_dataSet, test$data1sin) %>%
  arrange(test.data1sin)
ggplot(data = dfggplot)+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot[,1], color = "predict"))+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot[,2], color = "observe"))+
  labs(x = "data point", y = "sin value")

dfggplot2 <- data.frame(Fit_dataSet, test) %>%
  arrange(data1)
ggplot(data = dfggplot2)+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot2[,1], color = "predict"))+
  geom_point(aes(x = 1:length(Fit_dataSet),y = dfggplot2[,3], color = "observe"))+
  labs(x = "data point", y = "sin value")

h1_linear <- \(x) x
h2_ReLU <- \(x) ifelse(x >= 0, x, 0)
h3_softmax <- \(x) log(1+exp(x))

NNh1 <- neuralnet(data1sin~data1, data = train, hidden = 10, act.fct = h1_linear)
NNh2 <- neuralnet(data1sin~data1, data = train, hidden = 10, act.fct = h2_ReLU,
                  learningrate.limit = c(0,0.01))
# Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
NNh3 <- neuralnet(data1sin~data1, data = train, hidden = 10, act.fct = h3_softmax,
                  learningrate.limit = c(0,0.01))

fit_NNh1 <- predict(NNh1, newdata = test)
fit_NNh2 <- predict(NNh2, newdata = test)
fit_NNh3 <- predict(NNh3, newdata = test)

Q2dfggplot <- data.frame(test, fit_NNh1, fit_NNh2, fit_NNh3)
Q2dfggplot <- Q2dfggplot %>% arrange(data1)
ggplot(data = Q2dfggplot)+
  geom_point(aes(x = 1:length(fit_NNh1), y = data1sin, color = "observe"))+
  geom_point(aes(x = 1:length(fit_NNh1), y = fit_NNh1, color = "linear"))+
  geom_point(aes(x = 1:length(fit_NNh1), y = fit_NNh2, color = "ReLU"))+
  geom_point(aes(x = 1:length(fit_NNh1), y = fit_NNh3, color = "softmax"))+
  labs(x = "data point", y = "sin value")
```

```r
# Q2dfggplot <- Q2dfggplot %>% arrange(data1sin)
# ggplot(data = Q2dfggplot)+
#   geom_point(aes(x = 1:length(fit_NNh1), y = Q2dfggplot$data1sin, color = "observe"))+
#   geom_point(aes(x = 1:length(fit_NNh1), y = Q2dfggplot$fit_NNh1, color = "linear"))+
#   geom_point(aes(x = 1:length(fit_NNh1), y = Q2dfggplot$fit_NNh2, color = "ReLU"))+
#   geom_point(aes(x = 1:length(fit_NNh1), y = Q2dfggplot$fit_NNh3, color = "softmax"))
# # doesnt make sense
set.seed(1234567890)

data2 <- runif(500,0,50)
data2sin <- sin(data2)

dataSet2 <- tibble(data1 = data2, data1sin = data2sin)

sampleTrain <- sample(1:500,25)
train2 <- dataSet2[sampleTrain,]
test2 <- dataSet2[-sampleTrain,]

fit_Q3te <- predict(NNtrain, newdata = test2)

dfggplotQ3 <- data.frame(test2, fit_Q3te) %>% arrange(data1)

ggplot(data = dfggplotQ3)+
  geom_point(aes(x = 1:length(fit_Q3te), y = data1sin, color = "observe"))+
  geom_point(aes(x = 1:length(fit_Q3te), y = fit_Q3te, color = "predict"))+
  # geom_point(aes(x = 1:length(fit_Q3te), y = data1, color = "x"))+
  labs(x = "data point", y = "sin value")

NNtrain$weights
# sigmoid((c(1,9.10836573)%*% NNtrain$weights[[1]][[1]])) %*%
NNtrain$weights[[1]][[2]][2:11] + NNtrain$weights[[1]][[2]][1]
# plot(NNtrain)

sum(NNtrain$weights[[1]][[2]][which(NNtrain$weights[[1]][[1]][2,] >0) + 1,1]) +
  NNtrain$weights[[1]][[2]][1,1]

set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata
te <- mydata

NNQ5 <- neuralnet(Var~Sin, mydata, hidden = 10, threshold = 0.1)

plot(te[,2],predict(NNQ5,te), col="red", cex=1, type = "p",
     ylim = c(0,10), xlab = "Sin", ylab = "Var")
points(te[,2],te[,1], col = "blue", cex=1)
points(tr[,2], tr[,1],col = "black", cex=2)
```