# 932A99 - Group K7 - Lab 3

Hoda Fakharzadehjahromy, Otto Moen & Ravinder Reddy Atla

December 14, 2020

## Contents



Statement of contribution: Assignment 1 was contributed mostly by Ravinder, assignment 2 by Otto and assignment 3 by Hoda.

# 1. Assignment 1 - KERNEL METHODS

***1.Implement a kernel method to predict the hourly temperatures for a date and place in Sweden.To do so, you are provided with the files `stations.csv` and `temps50k.csv`. These files contain information about weather stations and temperature measurements in the stations at different days and times. The data have been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI).***

**1.1.a) You are asked to provide a temperature forecast for a date and place in Sweden. The**

forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels.

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

```
## Warning: package 'geosphere' was built under R version 4.0.3
```

```
# Cleaning Station data
stations <- read.table("stations.csv",sep=',')
Encoding(stations$V2) <- 'latin1'
colnames(stations) <- stations[1,]
stations <- stations[-1,]

temps <- read.csv("temps50k.csv")

# Merged stations and temperature data
st_org <- merge(stations,temps,by="station_number")
```

```
h_distance <- 102345
h_date <- 2345
h_time <- 10
a <- 58.4274
b <- 14.826
date <- "2013-11-04" # The date to predict (up to the students)
times <- c("04:00:00", "06:00:00",  "8:00:00", "10:00:00","12:00:00",
           "14:00:00", "16:00:00", "18:00:00", "20:00:00","22:00:00",
           "24:00:00")
```

```
filter_out_posterior <- function(st_org,date,time){
  filtered_data <- subset(st_org,st_org['date'] < date | st_org['date'] == date &
                            (st_org['time']) <= (time))
  return(filtered_data)
}
st <- filter_out_posterior(st_org,date,'08:00:00')
```

```
kernel_stations_dist <- function(my_data, a, b, h){
  coord_matrix <- as.matrix(my_data[,c(' longitude',' latitude')])
  mode(coord_matrix) <- 'numeric'
  coord_input_matrix <- matrix(rep(c(a,b),nrow(my_data)),ncol=2, byrow = TRUE)
  distance_station <- distHaversine(coord_matrix, coord_input_matrix, h)
  gaussian_kernel_dist <- exp(-1 * (distance_station/h) ^ 2)
  return(list(d = distance_station, k = gaussian_kernel_dist))
}
```

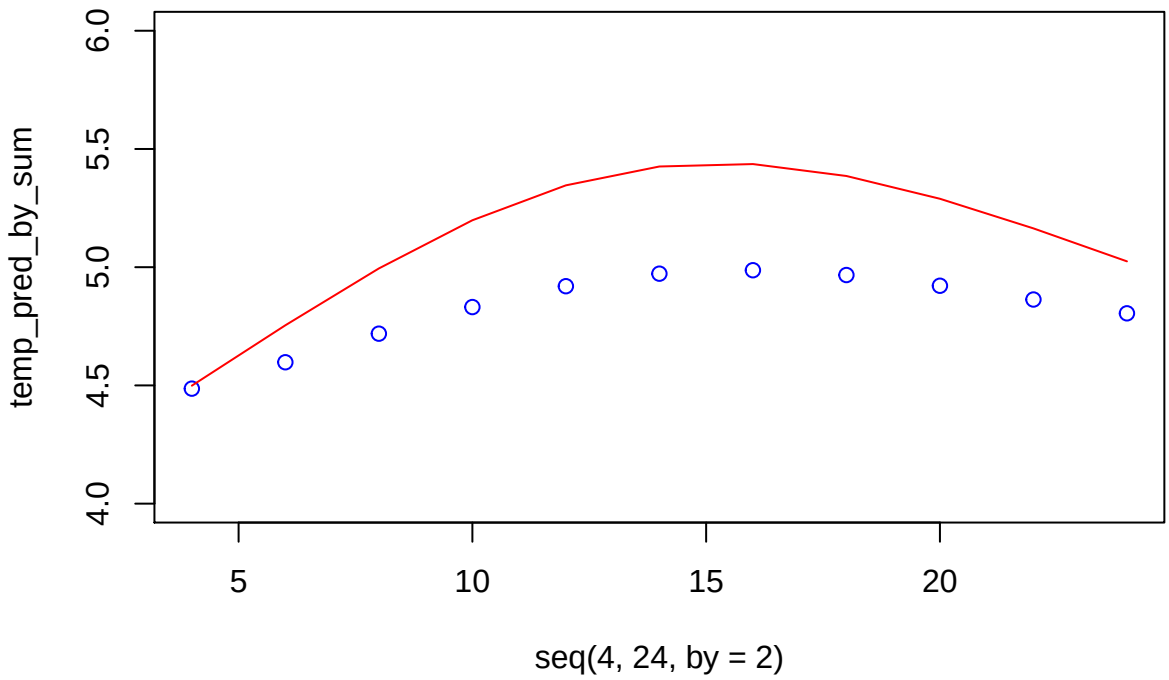
```r
kernel_day_dist <- function(my_data, date, h){
  date_from_data <- as.Date(my_data[,'date'])
  date_from_input <- as.Date(rep(date, nrow(my_data)))
  distance_days <- abs(as.numeric(difftime(date_from_data,date_from_input, units = 'days')))
  gaussian_kernel_days <- exp(-1 * ((distance_days)/h) ^ 2)
  return(list(d = distance_days, k = gaussian_kernel_days))
}
```

```r
kernel_time_dist <- function(time_data, time_input, h){
  time_data <- as.numeric(unlist(strsplit(time_data,':')))
  time_input <- as.numeric(unlist(strsplit(time_input,':')))

  #Time difference in hours
  time_data_hrs <- time_data[[1]] + (time_data[[2]]/60) + (time_data[[3]]/3600)
  time_input_hrs <- time_input[[1]] + (time_input[[2]]/60) + (time_input[[3]]/3600)
  distance_time <- round(abs(time_data_hrs - time_input_hrs))
  gaussian_kernel_time <- exp(-1 * (distance_time/h) ^ 2)
  return(list(d = distance_time,k = gaussian_kernel_time))
}
```

```r
temperature_forecast <- function(my_data,a,b,h_distance,h_date,h_time,
                                 date,times){
    stat_dist <- kernel_stations_dist(my_data,a,b,h_distance)[[2]]
    day_dist <- kernel_day_dist(my_data, date, h_date)[[2]]

    time_dist <- list()
    temperature_pred_sum <- vector()
    temperature_pred_prod <- vector()
    for(a_time in 1:length(times)){
      temp<- lapply(my_data[,'time'],
                                function(td) kernel_time_dist(td,times[a_time],
                                                              h_time))
      time_dist[[a_time]] <- unlist(temp)[seq(2, nrow(my_data) * 2, by = 2)]

      # Sum of Gaussian Kernel as kernel
      kernel_sum <- stat_dist + day_dist + time_dist[[a_time]]
      temperature_pred_sum[a_time] <- sum(kernel_sum * my_data[,'air_temperature']) /
        sum(kernel_sum)

      # Product of Gaussian Kernels as kernel
      kernel_prod <- stat_dist * day_dist * time_dist[[a_time]]
      temperature_pred_prod[a_time]<- sum(kernel_prod * my_data[,'air_temperature']) /
        sum(kernel_prod)
    }

    return(list(s = temperature_pred_sum, p = temperature_pred_prod))
}
```

```r
temp_pred_by_sum <- temperature_forecast(st,a,b,h_distance,h_date,h_time,
                                         date,times)[[1]]

temp_pred_by_prod <- temperature_forecast(st,a,b,h_distance,h_date,h_time,
                                         date,times)[[2]]
```

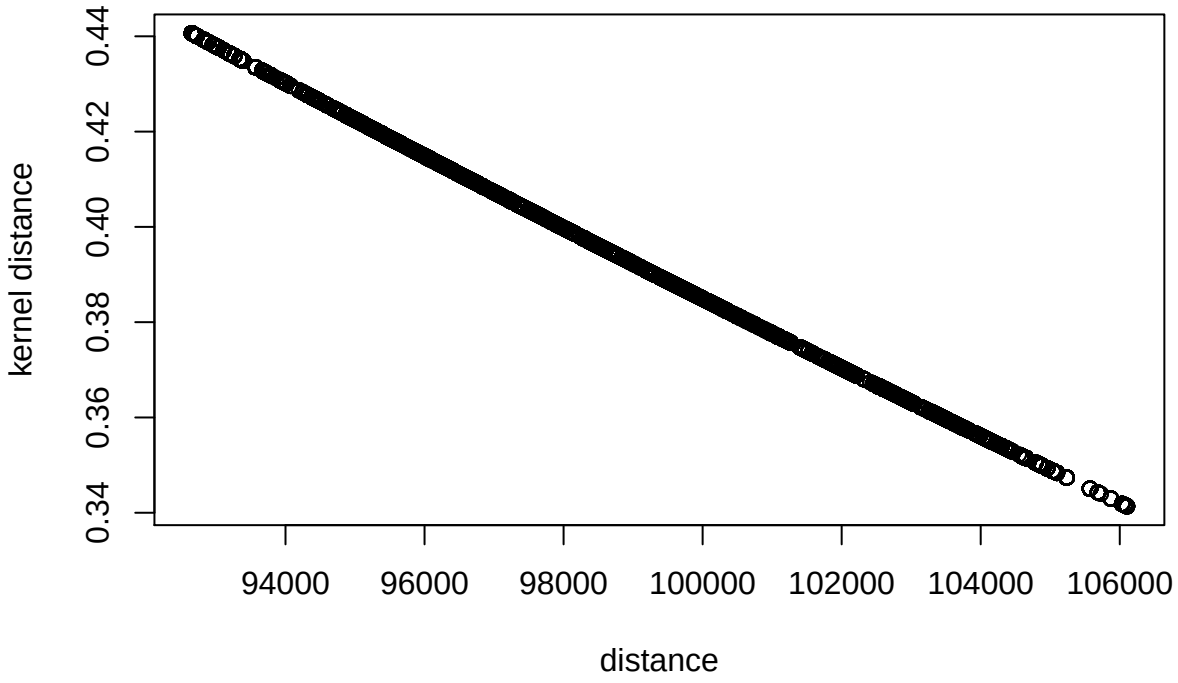
```
plot(x = seq(4,24,by=2), temp_pred_by_sum, col = 'blue',ylim = c(4,6))
lines(x = seq(4,24,by=2), temp_pred_by_prod, col = 'red')
```



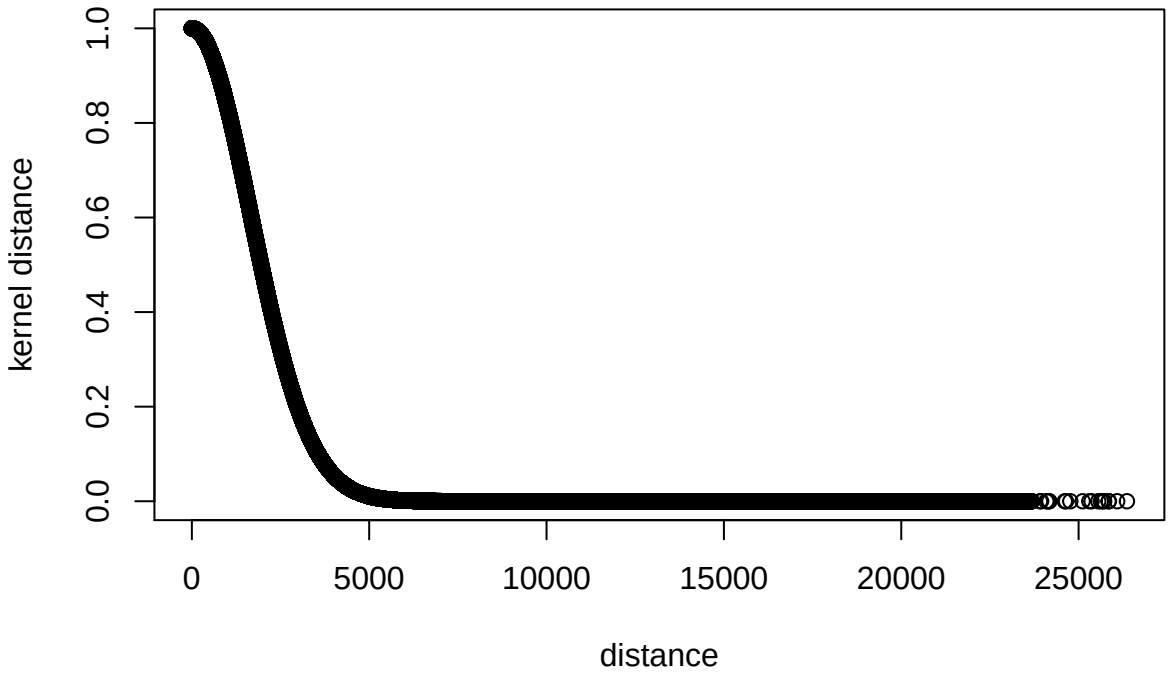

**1.2** ***Choose an appropriate smoothing coefficient or width for each of the three kernels above.No cross-validation should be used. Instead, choose manually a width that gives large kernel values to closer points and small values to distant points. Show this with a plot of the kernel value as a function of distance.Finally, repeat the exercise above by combining the three kernels into one by multiplying them, instead of summing them up. Compare the results obtained in both cases and elaborate on why they may differ.***

From the above plot, it is evident that temperatures predicted using sum of three Gaussian kernels is less than that of their product. This is because of the difference in the parameters used for predicting the temperature i.e weights and normalizer in the denominator. Weights and normalizer value for product is larger than for sum of kernels.

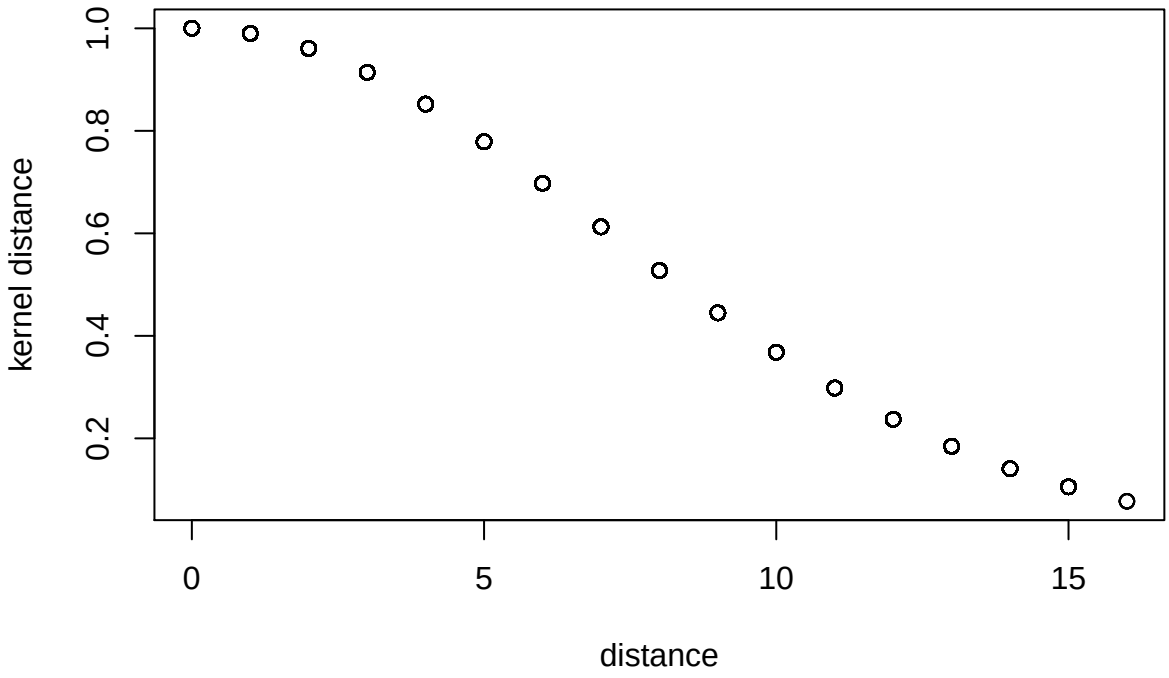
```
plot_station_dist <- plot(kernel_stations_dist(st,a,b,h_distance)[[1]],
                          kernel_stations_dist(st,a,b,h_distance)[[2]],
                          xlab = 'distance', ylab = 'kernel distance')
```




```
plot_day_dist <- plot(kernel_day_dist(st, date, h_date)[[1]],
                      kernel_day_dist(st, date, h_date)[[2]],
                      xlab = 'distance', ylab = 'kernel distance')
```




```
tim <- lapply(st[,'time'],
              function(td) kernel_time_dist(td,'08:00:00',
                                            h_time))
plot_time_dist <- plot(unlist(tim)[seq(1, nrow(st) * 2, by = 2)],
                       unlist(tim)[seq(2, nrow(st) * 2, by = 2)],
                       xlab = 'distance', ylab = 'kernel distance')
```




## 2. Assignment 2 - Support Vector Machines

*This assignments consists of running the code from the provided file* `Lab3Block1_2020_SVMs.R` *and answering questions about the models. The code from the file is displayed below:*

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

library(kernlab)
set.seed(1234567890)

data(spam)

index <- sample(1:4601)
tr <- spam[index[1:3000], ]
va <- spam[index[3001:3800], ]
trva <- spam[index[1:3800], ]
te <- spam[index[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)
```

```
  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)
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)
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)
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)
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3
```

### *Questison 1: Which filter do we return to the user? filter0, filter1, filter2 or filter3? Why?*

We want our final model, the one we return to the user, to be as good as possible. As such we want it to have access to as much data as possible so that it can use more information while training. Therefore the filter we return to the user should be filter3, as the parameter "data=spam" shows that it was trained on the entire data set.

### *Questison 2: What is the estimate of the generalization error of the filter returned? err0, err1, err2 or err3? Why?*

While one might think that since we're returning filter3 we should also return the error calculated on filter3, this would be incorrect. The idea behind the error is that it should show what sort of error we can expect when the model is used to predict new data, i.e. data that was not used to train the model, as this is going to be the situation when the model is implemented by the user. As filter3 was trained with all of our data, the test data was also included for training. Therefore an error calculated based on the test data will not be accurate but will instead likely underestimate the error of our model. The error that should be reported is err1. This is because filter1 was trained only on the training data and then tested on the test data. This error will likely be higher than err3, but will be a better reflection of how our filter will actually perform.

# 3. Assignment 3 - NEURAL NETWORKS

***3. 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 pairs are the data available to you. Use 25 of the 500 points for training and the rest for test.Use any number of layers and hidden units that you consider appropriate. You do not need to apply early stopping.***

***a)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.***

First, we initialized the weights randomly to almost-zero values ,in order to avoid updating them in the same way.

```
library(neuralnet)
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test
# Random initialization of the weights in the interval [-1, 1]
set.seed(1234567890)
winit <- runif(301, -1, 1)
set.seed(1234567890)
hidUnit <- c(6,4)
# Train a Neural Network
set.seed(1234567890)
nn <- neuralnet(Sin ~Var,data = tr,hidden=hidUnit,
                startweights = winit,learningrate = 0.001,
                threshold = 1e-4)
#plot(nn)
```

2 hidden layer has been used. 6 hidden unit in the first latent layer and 4 hidden unit in the second latent unit. The Error of the model is very small , 9.8e−5.

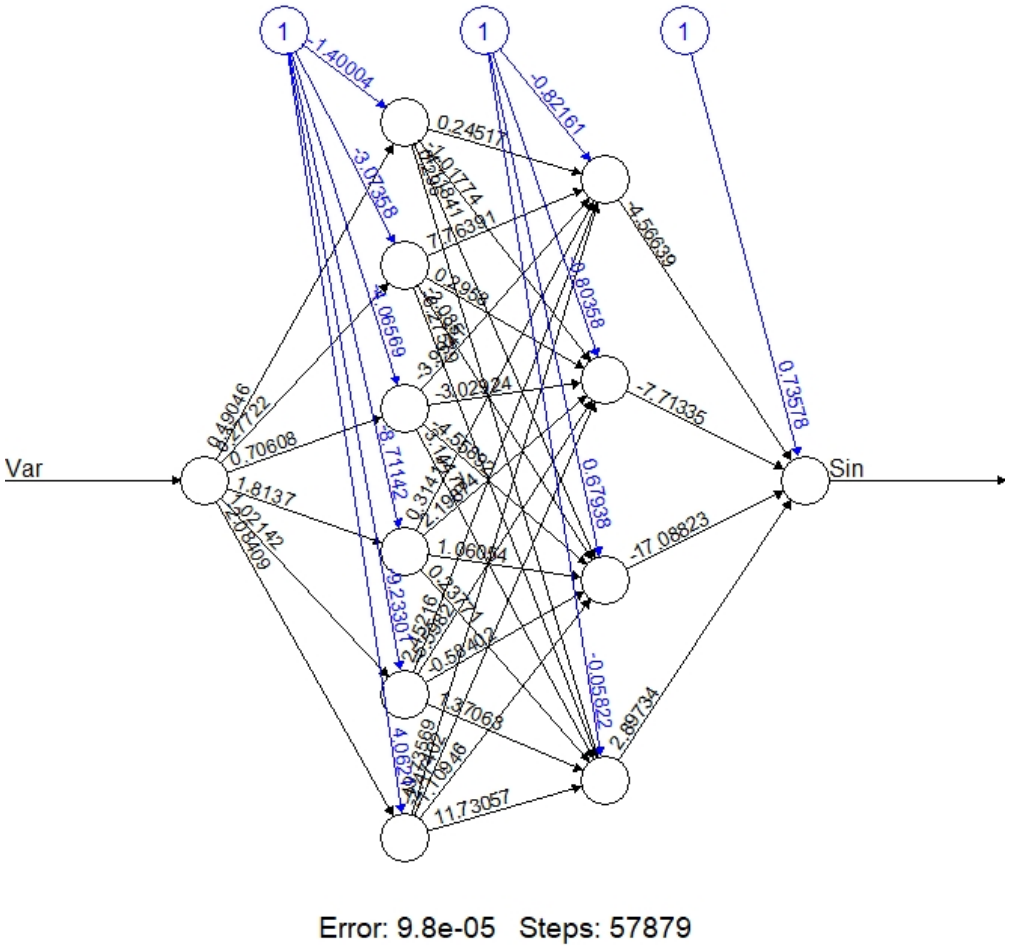
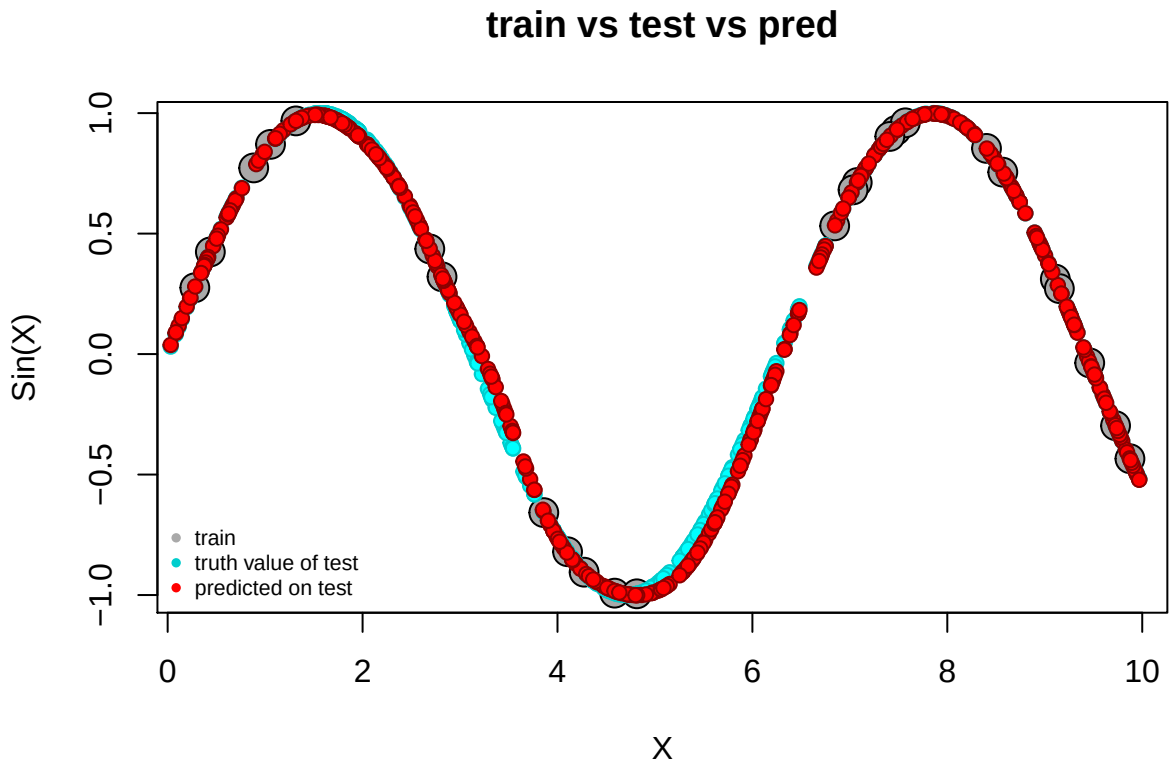



Figure 1: network architeture

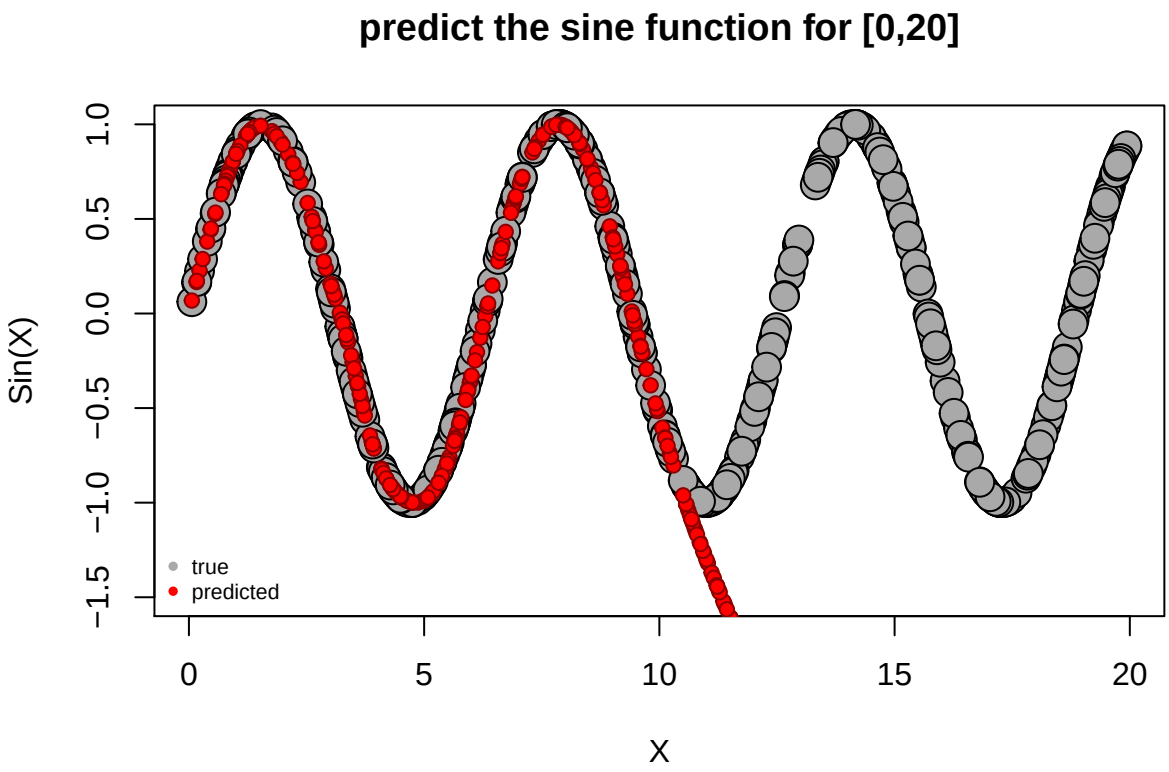
```
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2,main='train vs test vs pred',pch = 21,bg="darkgrey",
     xlab="X",ylab="Sin(X)")
points(te, col = "cyan3", cex=1,pch = 21,bg="cyan")
points(te[,1],predict(nn,te), col="darkred",bg="red", cex=1,pch=21)

legend("bottomleft", legend=c("train","truth value of test","predicted on test"),
       pch=c(16,16,16),col = c("darkgrey","cyan3","red"),
       cex = 0.65,bty = "n")
```




As seen in the graph, our model performs well, and predicts the test data with only a slight error.

***b) Then, sample 500 points uniformly at random in the interval*** $[0, 20]$***, and apply the sine function to each point. Use the previously learned NN to predict the sine function value for these new 500 points. You should get mixed results. Plot and comment your results.***

```
set.seed(1234567890)
Var2 <- runif(500, 0, 20)
mydata2 <- data.frame(Var=Var2, Sin=sin(Var2))
plot(mydata2, cex=2,main='predict the sine function for [0,20]',pch = 21,bg="darkgrey",
     xlab="X",ylab="Sin(X)",ylim=c(-1.5,1))
points(mydata2[,1],predict(nn,mydata2), col="darkred", cex=1,pch=21,bg="red")

legend("bottomleft", legend=c("true","predicted"), pch=c(16,16),col =
         c("darkgrey","red"),cex = 0.65,bty = "n")
```




From the graph, we can see that the our neural network has only been successful in predicting values of $x$ between $[0, 10]$ and clearly has not learned the general shape of $\sin x$ for $x$ between $[10, 20]$. This shows the network will not generalize to examples outside of the training set. Early stopping technique , could help the network avoid overfitting on the training data, which will result in a better generalization.

***c) Finally, 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 goals was to predict*** $\sin x$ ***from x. You should get bad results. Plot and comment your results***

The network architecture :

```
set.seed(1234567890)
Var3 <- runif(500, 0, 20)
mydata3 <- data.frame(Sin=sin(Var3),Var=Var3)
set.seed(1234567890)
winit <- runif(5500, -1, 1)
#hidUnit <- c(9,1)
set.seed(1234567890)
nn3 <-neuralnet(formula = Var~Sin,data = mydata3,
                hidden =c(4,2,1),startweights =winit,
               learningrate = 0.01,act.fct = "tanh")
```

In this task, the data to be learned is sequential and a feedforward neural network does not work for sequential or time-dependent data. Furthermore, in this example, we are trying to predict infinitely many x values from one sin(x) value. As a result, what we are trying to predict is not a function. A function maps every x value

to exactly one y value. In this case, there are theoretically infinitely many values that x can take on for every sin(x) we feed into the function. The domain of arcsin(x) is only from -1 to 1 and the range is from $-\pi/2$ to $\pi/2$ radians (not from 0 to 20). All these cause to poor prediction. One solution might be to constrain $x$ values to $-\pi/2$ to $\pi/2$

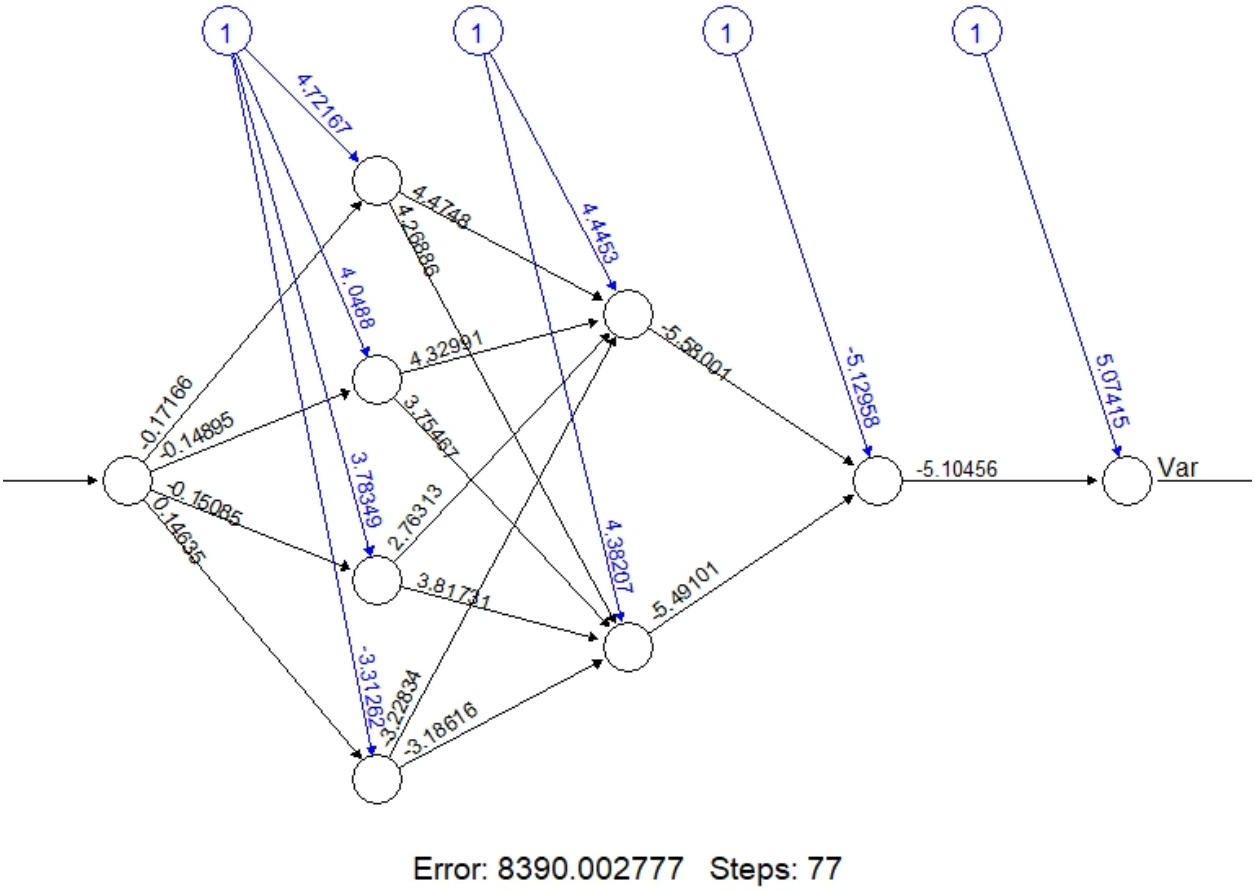
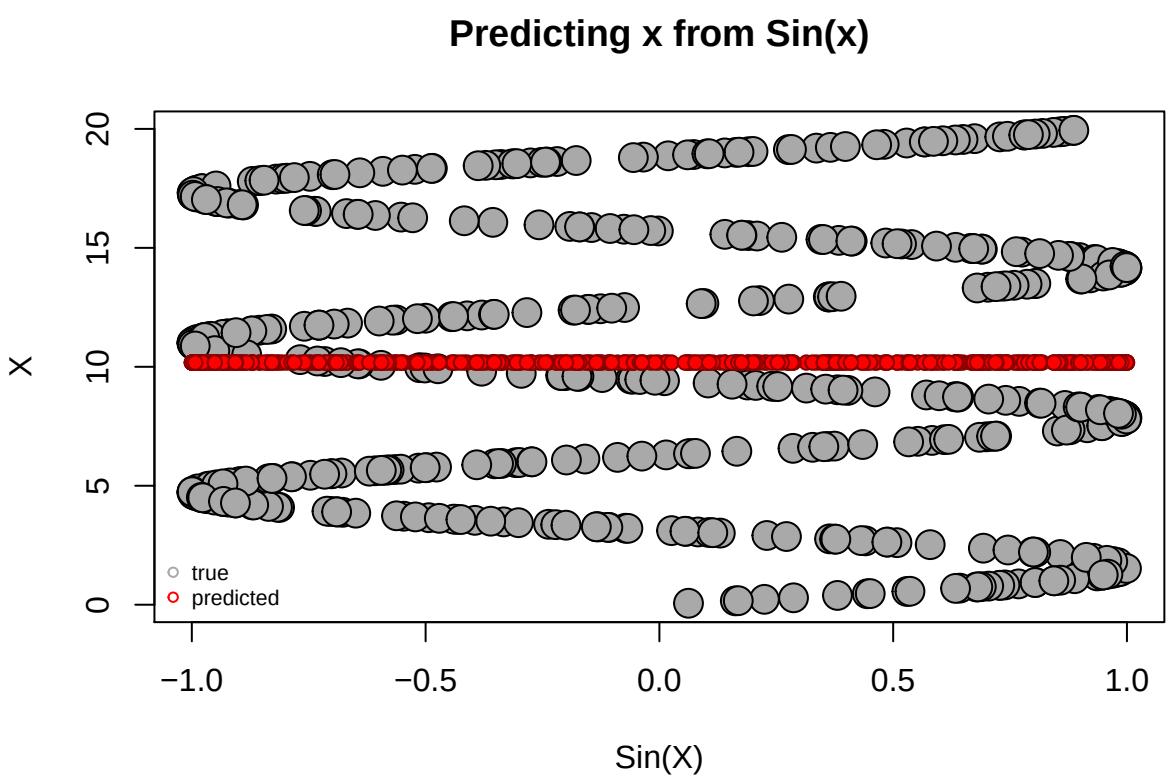



Figure 2: The network architecture

```
plot(mydata3, cex=2,main='Predicting x from Sin(x)',
     pch = 21,bg="darkgrey",
     ylab="X",xlab="Sin(X)")
points(mydata3[,1],predict(nn3,mydata3), col="darkred",
       cex=1,pch=21,bg="red")

legend("bottomleft", legend=c("true","predicted"), pch=c(21,21),
       col = c("darkgrey","red"),cex = 0.65,bty = "n")
```


Predicting x from Sin(x)
X
0
5
10
15
20
true
predicted
−1.0
−0.5
0.0
0.5
1.0
Sin(X)

# Appendix:

```
################################
#Assignment 1
################################
knitr::opts_chunk$set(echo = TRUE)
set.seed(1234567890)
library(geosphere)
# Cleaning Station data
stations <- read.table("stations.csv",sep=',')
Encoding(stations$V2) <- 'latin1'
colnames(stations) <- stations[1,]
stations <- stations[-1,]

temps <- read.csv("temps50k.csv")

# Merged stations and temperature data
st_org <- merge(stations,temps,by="station_number")

h_distance <- 102345
h_date <- 2345
h_time <- 10
a <- 58.4274
b <- 14.826
date <- "2013-11-04" # The date to predict (up to the students)
times <- c("04:00:00", "06:00:00",  "8:00:00", "10:00:00","12:00:00",
           "14:00:00", "16:00:00", "18:00:00", "20:00:00","22:00:00",
           "24:00:00")
filter_out_posterior <- function(st_org,date,time){
  filtered_data <- subset(st_org,st_org['date'] < date | st_org['date'] == date &
                            (st_org['time']) <= (time))
  return(filtered_data)
}
st <- filter_out_posterior(st_org,date,'08:00:00')

kernel_stations_dist <- function(my_data, a, b, h){
  coord_matrix <- as.matrix(my_data[,c(' longitude',' latitude')])
  mode(coord_matrix) <- 'numeric'
  coord_input_matrix <- matrix(rep(c(a,b),nrow(my_data)),ncol=2, byrow = TRUE)
  distance_station <- distHaversine(coord_matrix, coord_input_matrix, h)
  gaussian_kernel_dist <- exp(-1 * (distance_station/h) ^ 2)
  return(list(d = distance_station, k = gaussian_kernel_dist))
}


kernel_day_dist <- function(my_data, date, h){
  date_from_data <- as.Date(my_data[,'date'])
  date_from_input <- as.Date(rep(date, nrow(my_data)))
  distance_days <- abs(as.numeric(difftime(date_from_data,date_from_input, units = 'days')))
  gaussian_kernel_days <- exp(-1 * ((distance_days)/h) ^ 2)
  return(list(d = distance_days, k = gaussian_kernel_days))
}
```

```r
kernel_time_dist <- function(time_data, time_input, h){
  time_data <- as.numeric(unlist(strsplit(time_data,':')))
  time_input <- as.numeric(unlist(strsplit(time_input,':')))

  #Time difference in hours
  time_data_hrs <- time_data[[1]] + (time_data[[2]]/60) + (time_data[[3]]/3600)
  time_input_hrs <- time_input[[1]] + (time_input[[2]]/60) + (time_input[[3]]/3600)
  distance_time <- round(abs(time_data_hrs - time_input_hrs))
  gaussian_kernel_time <- exp(-1 * (distance_time/h) ^ 2)
  return(list(d = distance_time,k = gaussian_kernel_time))
}

temperature_forecast <- function(my_data,a,b,h_distance,h_date,h_time,
                                 date,times){
    stat_dist <- kernel_stations_dist(my_data,a,b,h_distance)[[2]]
    day_dist <- kernel_day_dist(my_data, date, h_date)[[2]]

    time_dist <- list()
    temperature_pred_sum <- vector()
    temperature_pred_prod <- vector()
    for(a_time in 1:length(times)){
      temp<- lapply(my_data[,'time'],
                                function(td) kernel_time_dist(td,times[a_time],
                                                              h_time))
      time_dist[[a_time]] <- unlist(temp)[seq(2, nrow(my_data) * 2, by = 2)]

      # Sum of Gaussian Kernel as kernel
      kernel_sum <- stat_dist + day_dist + time_dist[[a_time]]
      temperature_pred_sum[a_time] <- sum(kernel_sum * my_data[,'air_temperature']) /
        sum(kernel_sum)

      # Product of Gaussian Kernels as kernel
      kernel_prod <- stat_dist * day_dist * time_dist[[a_time]]
      temperature_pred_prod[a_time]<- sum(kernel_prod * my_data[,'air_temperature']) /
        sum(kernel_prod)
    }

    return(list(s = temperature_pred_sum, p = temperature_pred_prod))
}

temp_pred_by_sum <- temperature_forecast(st,a,b,h_distance,h_date,h_time,
                                         date,times)[[1]]

temp_pred_by_prod <- temperature_forecast(st,a,b,h_distance,h_date,h_time,
                                         date,times)[[2]]

plot(x = seq(4,24,by=2), temp_pred_by_sum, col = 'blue',ylim = c(4,6))
lines(x = seq(4,24,by=2), temp_pred_by_prod, col = 'red')
plot_station_dist <- plot(kernel_stations_dist(st,a,b,h_distance)[[1]],
                          kernel_stations_dist(st,a,b,h_distance)[[2]],
                          xlab = 'distance', ylab = 'kernel distance')
```

```r
plot_day_dist <- plot(kernel_day_dist(st, date, h_date)[[1]],
                      kernel_day_dist(st, date, h_date)[[2]],
                      xlab = 'distance', ylab = 'kernel distance')

tim <- lapply(st[,'time'],
              function(td) kernel_time_dist(td,'08:00:00',
                                            h_time))
plot_time_dist <- plot(unlist(tim)[seq(1, nrow(st) * 2, by = 2)],
                       unlist(tim)[seq(2, nrow(st) * 2, by = 2)],
                       xlab = 'distance', ylab = 'kernel distance')
################################
#Assignment 2
################################
# Lab 3 block 1 of 732A99/TDDE01 Machine Learning
# Author: jose.m.pena@liu.se
# Made for teaching purposes

library(kernlab)
set.seed(1234567890)

data(spam)

index <- sample(1:4601)
tr <- spam[index[1:3000], ]
va <- spam[index[3001:3800], ]
trva <- spam[index[1:3800], ]
te <- spam[index[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)
  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)
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)
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)
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)
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3
################################
#Assignment 3
################################
library(neuralnet)
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test
# Random initialization of the weights in the interval [-1, 1]
set.seed(1234567890)
winit <- runif(301, -1, 1)
set.seed(1234567890)
hidUnit <- c(6,4)
# Train a Neural Network
set.seed(1234567890)
nn <- neuralnet(Sin ~Var,data = tr,hidden=hidUnit,
                startweights = winit,learningrate = 0.001,
                threshold = 1e-4)
#plot(nn)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2,main='train vs test vs pred',pch = 21,bg="darkgrey",
     xlab="X",ylab="Sin(X)")
points(te, col = "cyan3", cex=1,pch = 21,bg="cyan")
points(te[,1],predict(nn,te), col="darkred",bg="red", cex=1,pch=21)

legend("bottomleft", legend=c("train","truth value of test","predicted on test"),
       pch=c(16,16,16),col = c("darkgrey","cyan3","red"),
       cex = 0.65,bty = "n")


set.seed(1234567890)
Var2 <- runif(500, 0, 20)
mydata2 <- data.frame(Var=Var2, Sin=sin(Var2))
plot(mydata2, cex=2,main='predict the sine function for [0,20]',pch = 21,bg="darkgrey",
     xlab="X",ylab="Sin(X)",ylim=c(-1.5,1))
points(mydata2[,1],predict(nn,mydata2), col="darkred", cex=1,pch=21,bg="red")

legend("bottomleft", legend=c("true","predicted"), pch=c(16,16),col =
         c("darkgrey","red"),cex = 0.65,bty = "n")
set.seed(1234567890)
Var3 <- runif(500, 0, 20)
mydata3 <- data.frame(Sin=sin(Var3),Var=Var3)
set.seed(1234567890)
winit <- runif(5500, -1, 1)
```

```
#hidUnit <- c(9,1)
set.seed(1234567890)
nn3 <-neuralnet(formula = Var~Sin,data = mydata3,
                hidden =c(4,2,1),startweights =winit,
               learningrate = 0.01,act.fct = "tanh")


plot(mydata3, cex=2,main='Predicting x from Sin(x)',
     pch = 21,bg="darkgrey",
     ylab="X",xlab="Sin(X)")
points(mydata3[,1],predict(nn3,mydata3), col="darkred",
       cex=1,pch=21,bg="red")

legend("bottomleft", legend=c("true","predicted"), pch=c(21,21),
       col = c("darkgrey","red"),cex = 0.65,bty = "n")
```