Lab 3 TDDE01

Filip Cornell

2018-12-11

1 Assignment 1 - The kernel trick

In this task, we are asked to perform predictions for a certain location in Sweden between the hours 04:00:00 and 00:00:00 on a specific date. I chose Norrkoping on the 11th of November 2013.

We were asked to to do this using three kernels; one for the distance between the cities, one for the day and one for the time of the day. The first task was to set the width parameters. I chose the parameters as follows.

```
h_distance <- 100000 # reasonable that it is the same within 200km more or less
h_date <- 10 # If the year was extra cold. Reasonable to have within one year.
h_time <- 2 # About an hour ish. Reasonable.
```

I consider it reasonable that we have more or less the same climate within a radius of 100 km, which is why I chose `h_distance` as above. I chose `h_date` as 180, one unit is one day, and I believe that same years are colder than others, thus yielding a colder temperature overall, which I want to yield signifiance then. The parameter`h_time` I chose to be 4, as one unit is about 15 minutes.

After this, we initialize all the necessary functions and commence the problem.

```
set.seed(1234567890)

library(geosphere)
stations <- read.csv("stations.csv",
                     stringsAsFactors=FALSE,
                     fileEncoding="latin1")
temps <- read.csv("temps50k.csv")
# Creating Month and Day of year.
temps$MonthDay = factor(substr(as.character(temps$date),
                               start = 6,
                               stop = 10))
st <- merge(stations,temps,by="station_number")

date <- "2013-11-04" # The date to predict (up to the students)
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", "00:00:00")
temp <- vector(length=length(times))
monthDay = "11-04"
# My own code
# Can take in a vector aswell. Good.
# gaussianKernel = function(x1,x2, h) {
#   u = (x1 - x2)/h
#   return(exp(-(u^2)))
# }

convert_to_mins = function(time) {

  return(as.numeric(substr(time,1,2))*60 + as.numeric(substr(time,4,5)))
```

```r
}


gaussianDateKernel = function(day1, day2, h) {
  if (length(day2 > 1)) {
    daysBetween = apply(as.matrix(day2), 1, function(d2) {
      min(366 - abs(day1 - d2), abs(day1 - d2))
    })
  } else {
    daysBetween = min(366 - abs(day1 - day2), abs(day1 - day2))
  }
  return(exp(-(daysBetween^2)/(h^2)))
}

gaussianDistKernel = function(dist, h) {
  return(exp(-(dist^2)/(h^2)))
}

# Time received in minutes
timeKernel = function(time1, time2, h) {
  timeDiff = abs((time1 - time2)/60)
  #print(timeDiff)

  if (length(timeDiff) > 1) {
    timeDiff = apply(as.matrix(timeDiff), 1, function(row) {
      if (row > 12) {
        return(12 - row %% 12)
      } else {
        return(row)
      }
    })
  }

  return(exp(-(timeDiff^2)/(h^2)))
}


sumKernel = function(dist1, dist2, date1, date2, time1, time2) {
  return(gaussianDistKernel(dist1, dist2, h_distance) +
           gaussianDateKernel(date1, date2, h_date) +
           timeKernel(time1, time2, h_time))
}

prodKernel = function(dist1, dist2, date1, date2, time1, time2) {
  return(gaussianDistKernel(dist1, dist2, h_distance) *
           gaussianDateKernel(date1, date2, h_date) *
           timeKernel(time1, time2, h_time))
}

# Preprocess data to use

featuresToUse = data.matrix(st[,-c(1,2,3,6,7)])
# Set time to minutes
```

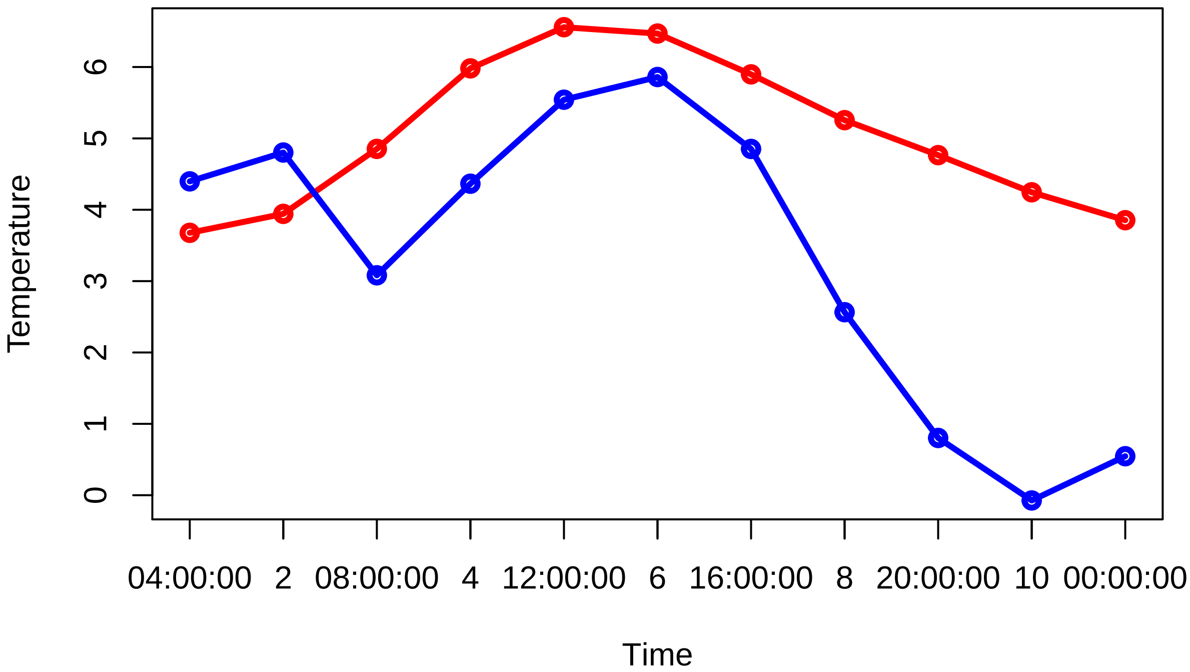
```r
featuresToUse[,5] = apply(as.matrix(as.character(st$time)),1,convert_to_mins)#convert_to_mins(st$time)
featuresToUse = featuresToUse[,-c(3,7)]
featuresToUse[,1] = as.numeric(featuresToUse[,1])
featuresToUse[,2] = as.numeric(featuresToUse[,2])

# For sensitivity; show distance between
# Norrkoping and linkoping and different hours and different days

get_points = function(stationName, date, times, monthDay) {
  point = matrix(NA, nrow = length(times), ncol = 5)
  point[,1] = rep(as.numeric(stations[stations$station_name == stationName,]$latitude),
                times = length(times))
  point[,2] = rep(as.numeric(stations[stations$station_name == stationName,]$longitude),
                times = length(times))
  point[,3] = rep(factor(date,
                         levels = levels(st$date)),
                times = length(times))
  point[,4] = apply(as.matrix(times), 1, convert_to_mins)

  point[,5] = factor(monthDay,
                     levels = levels(temps$MonthDay))
  colnames(point) = c(colnames(featuresToUse)[1:4],colnames(featuresToUse)[6])

  return(point)
}

pred_point_sum_kernel = function(pointToPred, data) {
  data = data[data[,3] <= pointToPred[3],]
  temps_data = as.matrix(data[,5])
  distances = apply(data[,1:2], 1, function(dataCoords) {
    return(distHaversine(dataCoords, pointToPred[1:2]))
  })
  kernSum = as.matrix(timeKernel(time1 = rep(pointToPred[4], times = nrow(data)),
                                  time2 = data[,4], h = h_time) +
    gaussianDistKernel(dist = distances,
                       h = h_distance) +
    gaussianDateKernel(day1 = rep(pointToPred[3],
                       times = nrow(data)),
                     day2 = data[,3],
                     h = h_date))
  above = (t(kernSum)%*%temps_data)[1,1]
  below = sum(kernSum)
  return(above/below)
}

pred_point_prod_kernel = function(pointToPred, data) {
  data = data[data[,3] <= pointToPred[3],]
  temps_data = as.matrix(data[,5])

  distances = apply(data[,1:2], 1, function(dataCoords) {
    return(distHaversine(dataCoords, pointToPred[1:2]))
  })
  kernProd = as.matrix(timeKernel(time1 = rep(pointToPred[4], times = nrow(data)),
```

```
                                   time2 = data[,4],
                                   h = h_time) *
                        gaussianDistKernel(dist = distances,
                                        h = h_distance) *
                        gaussianDateKernel(day1 = rep(pointToPred[3], times = nrow(data)),
                                    day2 = data[,3],
                                    h = h_date))
  above = (t(kernProd)%*%temps_data)[1,1]
  below = sum(kernProd)
  return(above/below)
}
```

1.1 Predicting temperature for Norrkoping

As mentioned previously, I chose Norrkoping to predict for. This can be seen below.

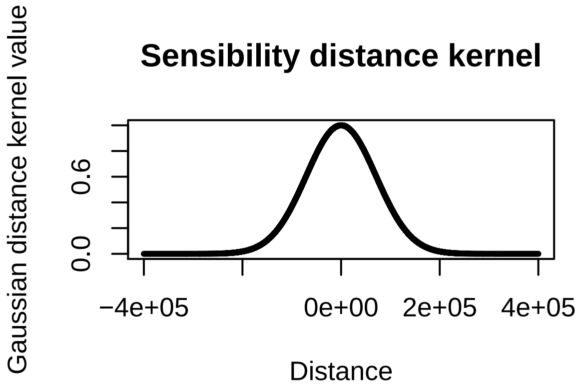
```
norrkopingTimes = get_points(stationName = "Norrköping",
                             date = date,
                             times = times,
                             monthDay = monthDay)
LinkopingTimes = get_points(stationName = "Linköping",
                            date = date,
                            times = times,
                            monthDay = monthDay)


# Calculate temp
tempsSum = as.numeric()
tempsProd = as.numeric()
#count = 0
for (i in 1:nrow(norrkopingTimes)) {
  tempsSum[i] = pred_point_sum_kernel(norrkopingTimes[i,], data = featuresToUse)
  tempsProd[i] = pred_point_prod_kernel(norrkopingTimes[i,], data = featuresToUse)
  #count = count + 1
  #print(count)


}

# Plot
plot(tempsSum, type="o",
     ylim = c(min(tempsSum,tempsProd), max(tempsSum,tempsProd)),
     col = "red", lwd = 3, xlab = "Time", ylab = "Temperature")
lines(tempsProd, type = "o", col = "blue", lwd = 3)
axis(1, at=1:11, labels=times)
```

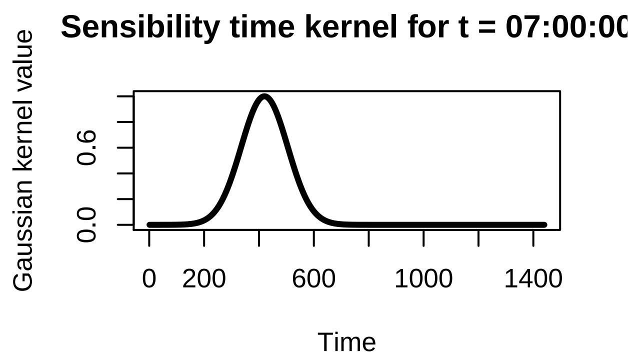


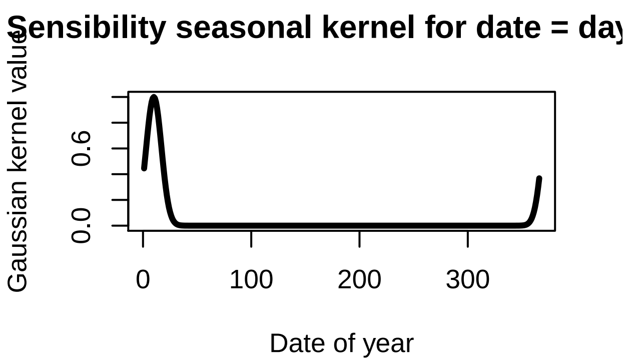
1.2 Showing sensitivity

We were asked to show that our kernels are sensitive. Thus, I here generate a few plots to show how each of the kernels individually are sensitive.

```
par(mfrow=c(2,2))
distSeq = seq(-400000, 400000, 40)
plot(x=distSeq,
     gaussianDistKernel(dist = distSeq, h = h_distance),
     type = "l", lwd = 3,
     xlab = "Distance",
     ylab = "Gaussian distance kernel value",
     main = "Sensibility distance kernel")
# Show for time

timeSeq = seq(1,1440,1)
plot(x=timeSeq, y = timeKernel(time1 = 60*7, time2 = timeSeq, h = h_time),
     type = "l",
     lwd = 3,
     xlab = "Time",
     ylab = "Gaussian kernel value",
     main = "Sensibility time kernel for t = 07:00:00")

dateSeq = seq(1,366,1)
plot(x=dateSeq,
     y = gaussianDateKernel(day1 = 10, day2 = dateSeq, h = h_date),
     type = "l", lwd = 3, xlab = "Date of year",
     ylab = "Gaussian kernel value",
     main = "Sensibility seasonal kernel for date = day 10")
par(mfrow=c(1,1))
```







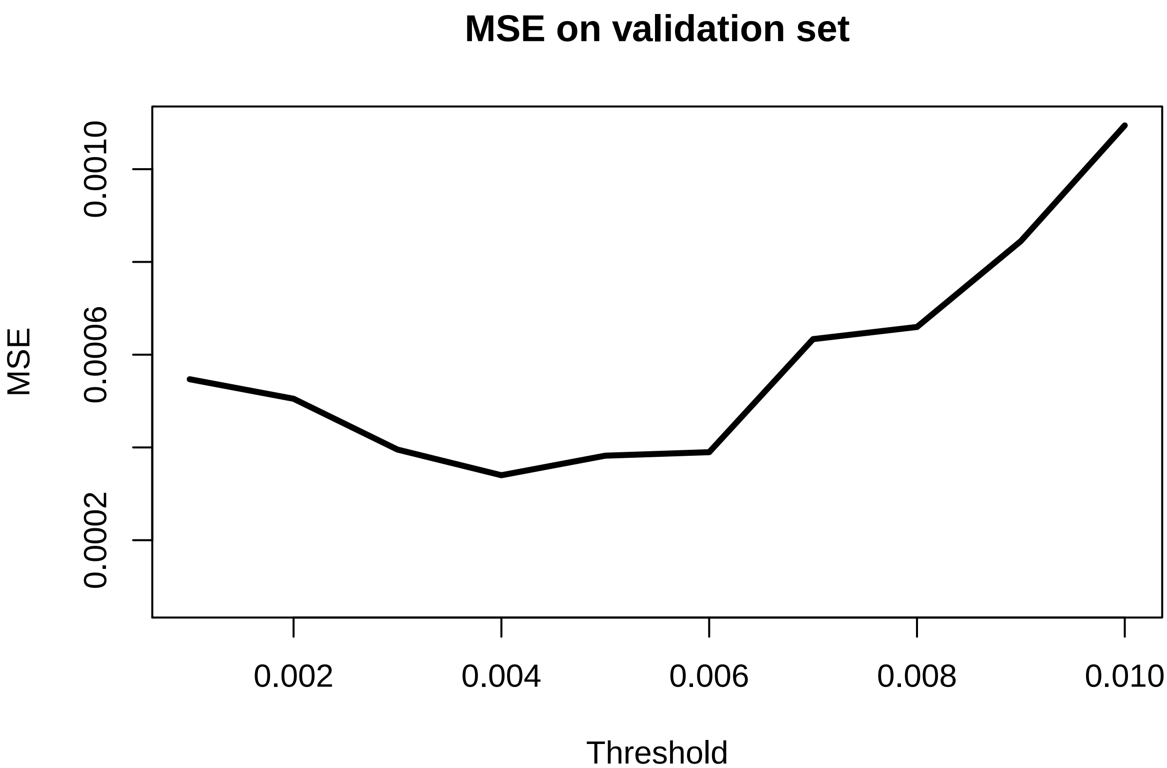
We can clearly see that the kernels are less sensitive the longer the distance from their actual values. Thus, we have shown their sensitivity.

2 Neural networks

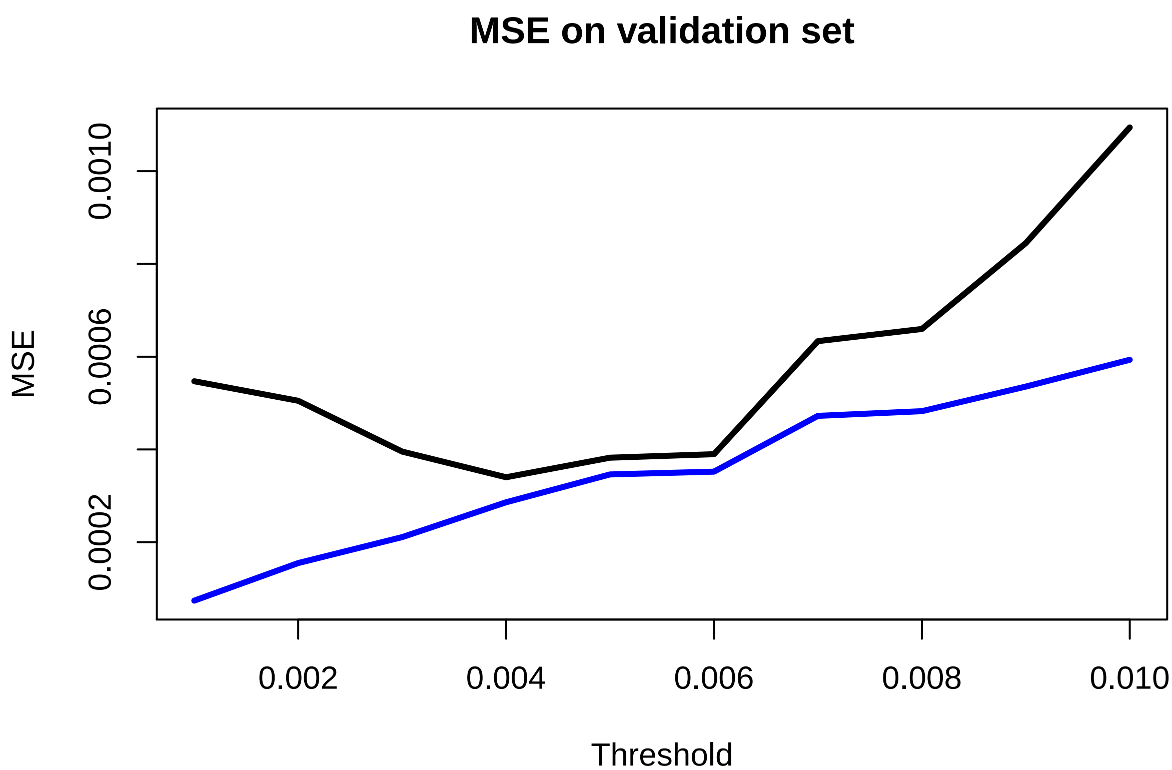
In this task, we were asked to implement a neural network with 10 hidden layers. To obtain the optimal neural network, we try with different thresholds and use the optimal neural network with the threshold that yields the smallest MSE on the predictions. We arrive at the conclusion the threshold 0.004 is optimal on the validation, and we can see in the plot the it approximates the function very nicely.

```
# Reuse function created in lab 1
MSE = function(y, y_hat) {
  n = length(y)
  return((1/n)*sum((y - y_hat)^2))
}

#install.packages("neuralnet")
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(50, min = -1, max = 1)

thr = as.numeric()
```

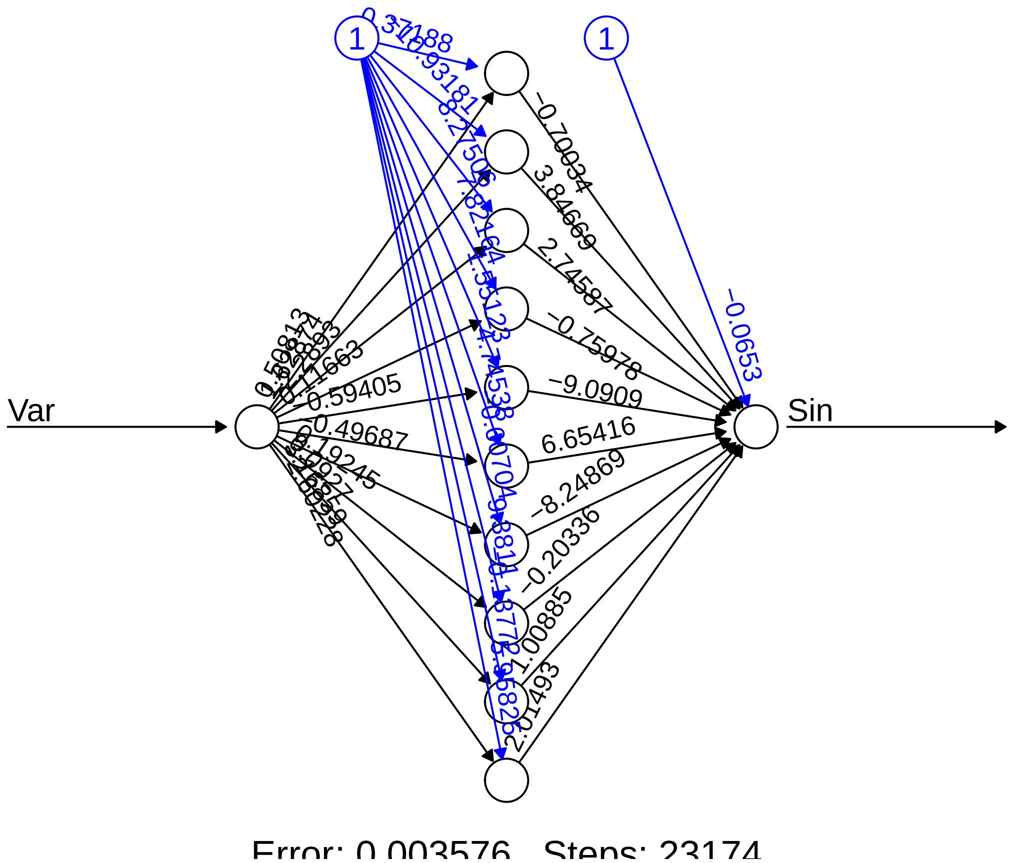
```r
MSEsValidation = as.numeric()
MSEsTrain = as.numeric()
for (i in 1:10) {
    thr[i] = i/1000
    nn <- neuralnet(Sin ~ Var, data = tr,
                    threshold = thr[i], startweights = winit, hidden = 10)
    # Use validation set here to get error
    yValidation = compute(nn, va$Var)$net.result
    MSEsValidation[i] = MSE(y = va$Sin, y_hat = yValidation[,1])
    yTrain = compute(nn, tr$Var)$net.result
    MSEsTrain[i] = MSE(y = tr$Sin, y_hat = yTrain[,1])
}


plot(x = thr, y = MSEsValidation, type = "l",
     lwd = 3, xlab = "Threshold",
     ylab = "MSE", main = "MSE on validation set",
     ylim = c(min(MSEsTrain, MSEsValidation), max(MSEsTrain, MSEsValidation)))
```



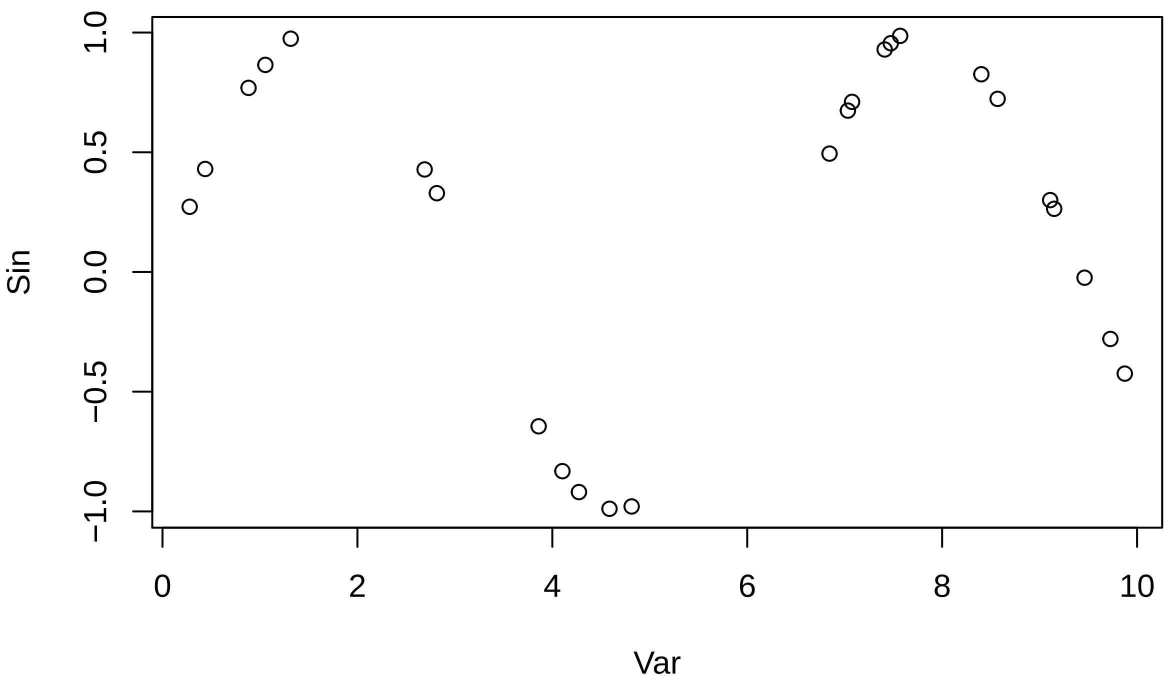

```r
lines(x = thr, y = MSEsTrain, col = "blue", lwd = 3)
```




```r
# Choose the threshold giving the least amount of error.
bestThr = thr[which.min(MSEsValidation)]

# Plot the best one
plot(nn <- neuralnet(Sin ~ Var, data = tr, threshold = bestThr, startweights = winit, hidden = 10), rep
```




```
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1)
```



```r
points(trva, col = "red")
```