Machine Learning Lab 3

Theodor Emanuelsson (theem089), Lisa Goldschmidtböing (lisgo269), Christoforos Spyretos (chrsp415)

2021-12-15

Assignment 1: Kernel Methods

```
set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")

#all data together
df <- merge(stations,temps,by="station_number")
```

At first, we define the place and date we will make the prediction for and delete all data observed later than our picked day. Then, we choose a day where we have a prediction in the dataset at a station close to our location to compare whether our estimation makes sense. Our estimation was for Tyrislöt on the 29th of January 1990.

```
# date we are choosing (can make a comparison to Holma station
# later and we need to define the date now for filtering data)
date <- as.Date("1990-01-29")

# making date in the dataset a date.object to use difftime to filter out all
# data from this day and after
df$date <- as.Date(df$date)
oldDf <- df #later comparison
df <- df[-which(difftime(date, df$date) <= 0),]
```

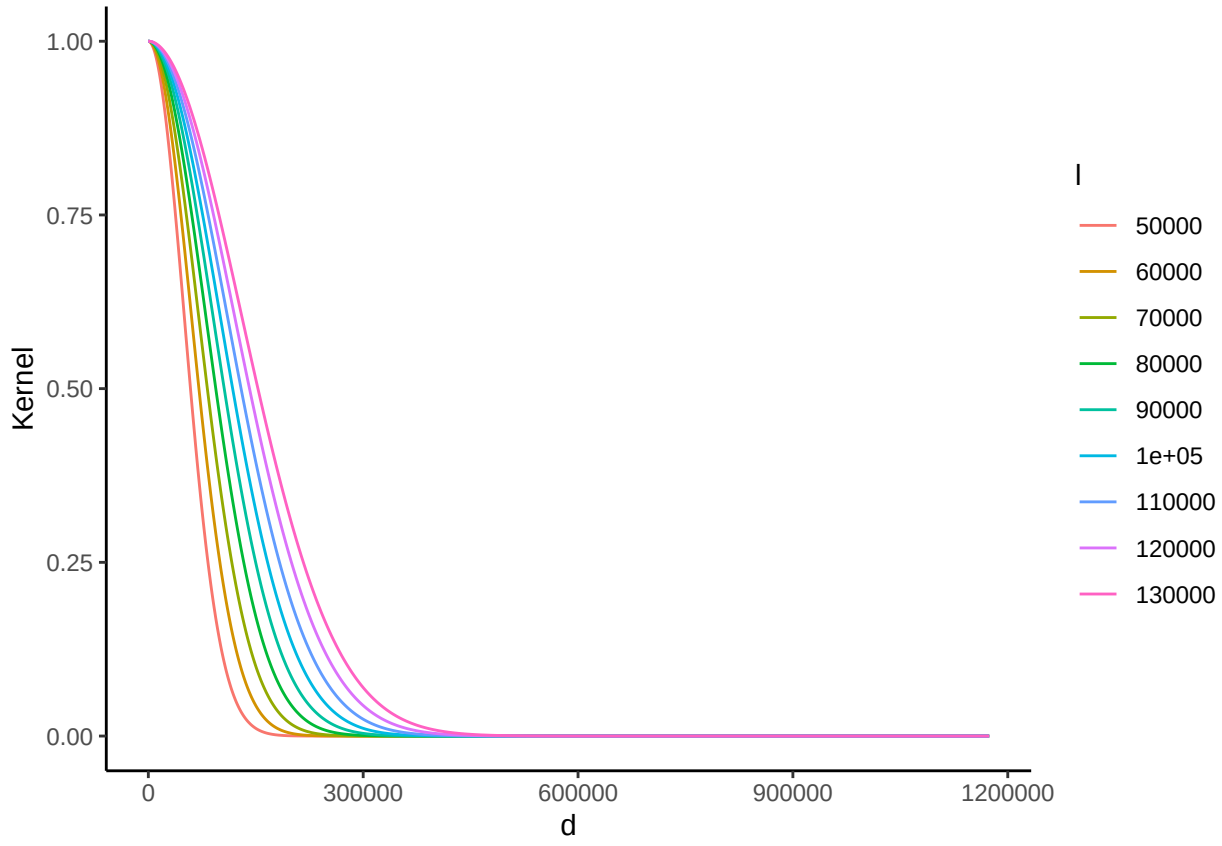
In the next step, we define our different kernels step by step and combine them afterwards. Within each step, we also need to find appropriate width coefficients. Some explanation about the procedure is given for each kernel.

The first kernel to account for the physical distance from a station to the point of interest.

First, we get our desired coordinates from google maps and then calculate the distance using the *distHaversine()* function. Next, we plot different width values between 50000 and 130000. We pick $l = 100000$ even though this is a quite random choice as many numbers around that area would have been a reasonable choice as well. We picked it for the reason that all points with a distance greater than 300km from our observation have close to 0 weight now, and that seemed reasonable to us.

```
# place we are going to pick is Tyrislöt (points from google maps)
tyris <- c(16.89461,58.32633)

# calculate distance
```

```r
d1 <- sapply(1:nrow(df), function(x) distHaversine(tyris,
                                                   c(df[x,]$longitude,
                                                     df[x,]$latitude)))

# plot different values for l
# x axis from zero up to maximal difference (max(d1))
h1 <- seq(0,max(d1),by=1000)

# how many different values do we wanna try and which sequence
m1 <- 9
mySeq1 <- seq(50000,130000,length.out=m1)

# calculate kernel for different values of l
results1 <- lapply(mySeq1, function(l) exp(-h1^2/(2*l^2)))
plot_df1 <- data.frame("Kernel"=unlist(results1),
                       "l"=rep(mySeq1,
                               each=length(h1)),
                       "d"=rep(h1, times=length(m1)))
plot_df1$l <- as.factor(plot_df1$l)
ggplot(plot_df1, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()
```




```r
# pick 100000
# apply to observations
```

```
k1 <- exp(-d1^2/(2*100000^2))
```

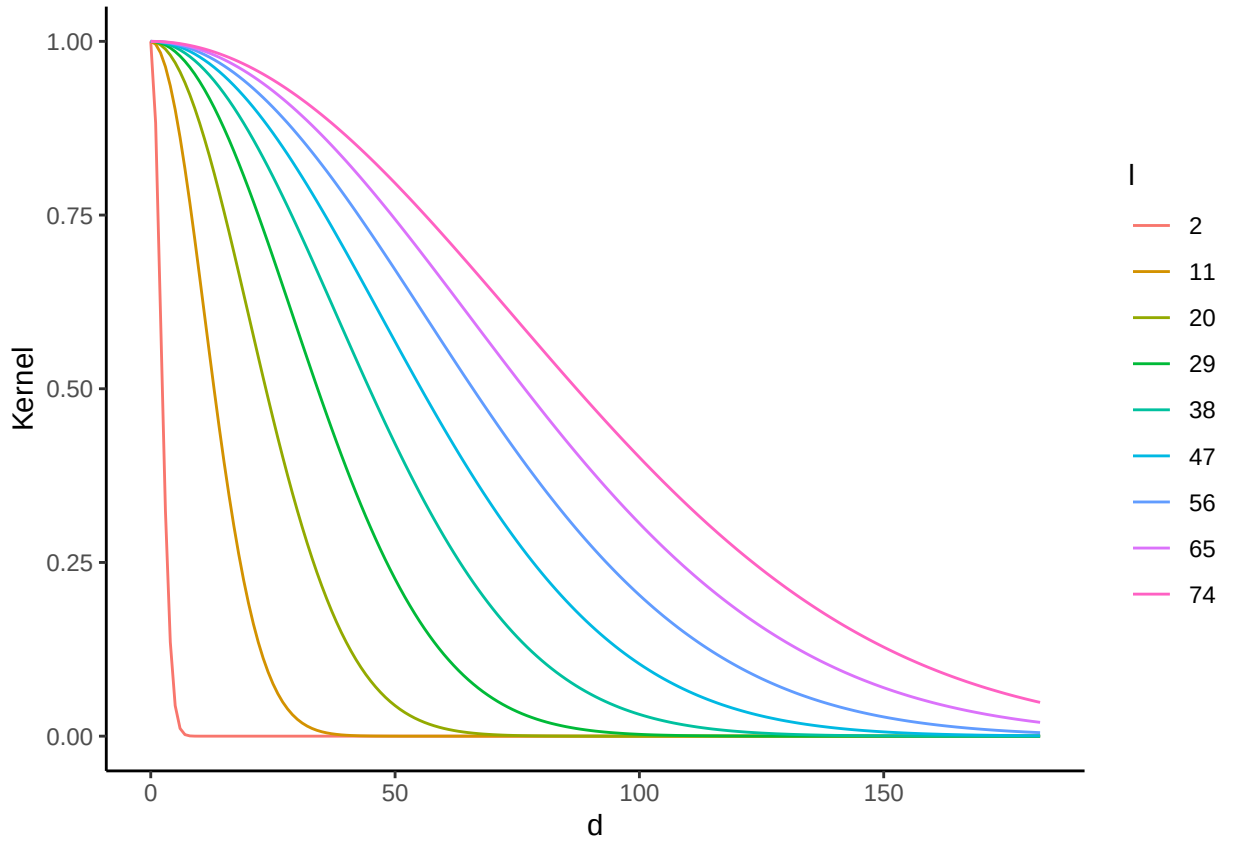
The second kernel to account for the distance between the day a temperature measurement was made and the day of interest.

First, we have to decide whether we make a total difference of the day to previous measures or compare within one year and ignore which year the measure was taken. We decide that the second attempt would be more reasonable as it considers seasons in a better way. Doing this, we might ignore the climate change impact over the years, which would be a reason for considering the total day difference. However, as this probably has a quite small impact on the few measurements, we have decided that the difference within a year would be more reasonable.

Using modulo and dividing by the total number of days per year, we receive the difference in days within one year. Considering that we have leap years every four years, we use 365.25 instead of 365 to get a better approximation. We could also round the results we achieve, only having full days as differences but decide to stick to the not rounded numbers as we do not need exact numbers and are more precise using these numbers. Additionally, one must keep in mind that we have the situation that December and January are right after each other in the same way as December and November are. Hence, we need to consider that there are always two "directions" in which the time difference can be calculated. Whenever our differences exceed half a year (182.625 days), we go the other way, which is shorter then and subtract the difference we calculated from the total number of days.

For this kernel, we pick $l = 11$ as all observations that are more than a month away get around zero weight.

```
# date we are choosing is the 29th of January 1990

# calculate difference
diff <- as.numeric(difftime(date, df$date)) %% 365.25
d2 <- ifelse(diff <= 182.625, diff, 365.25-diff)

# plot different values for l
# x axis from zero up to maximal difference (max(d2))
h2 <- seq(0,max(d2),by=1)

# how many different values do we wanna try and which sequence
m2 <- 9
mySeq2 <- seq(2,74,length.out=m2)

# calculate kernel for different values of l
results2 <- lapply(mySeq2, function(l) exp(-h2^2/(2*l^2)))
plot_df2 <- data.frame("Kernel"=unlist(results2),
                       "l"=rep(mySeq2,
                               each=length(h2)),
                       "d"=rep(h2, times=length(m2)))

plot_df2$l <- as.factor(plot_df2$l)
ggplot(plot_df2, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()
```




```
# pick 11
# apply to observations
k2 <- exp(-d2^2/(2*11^2))
```

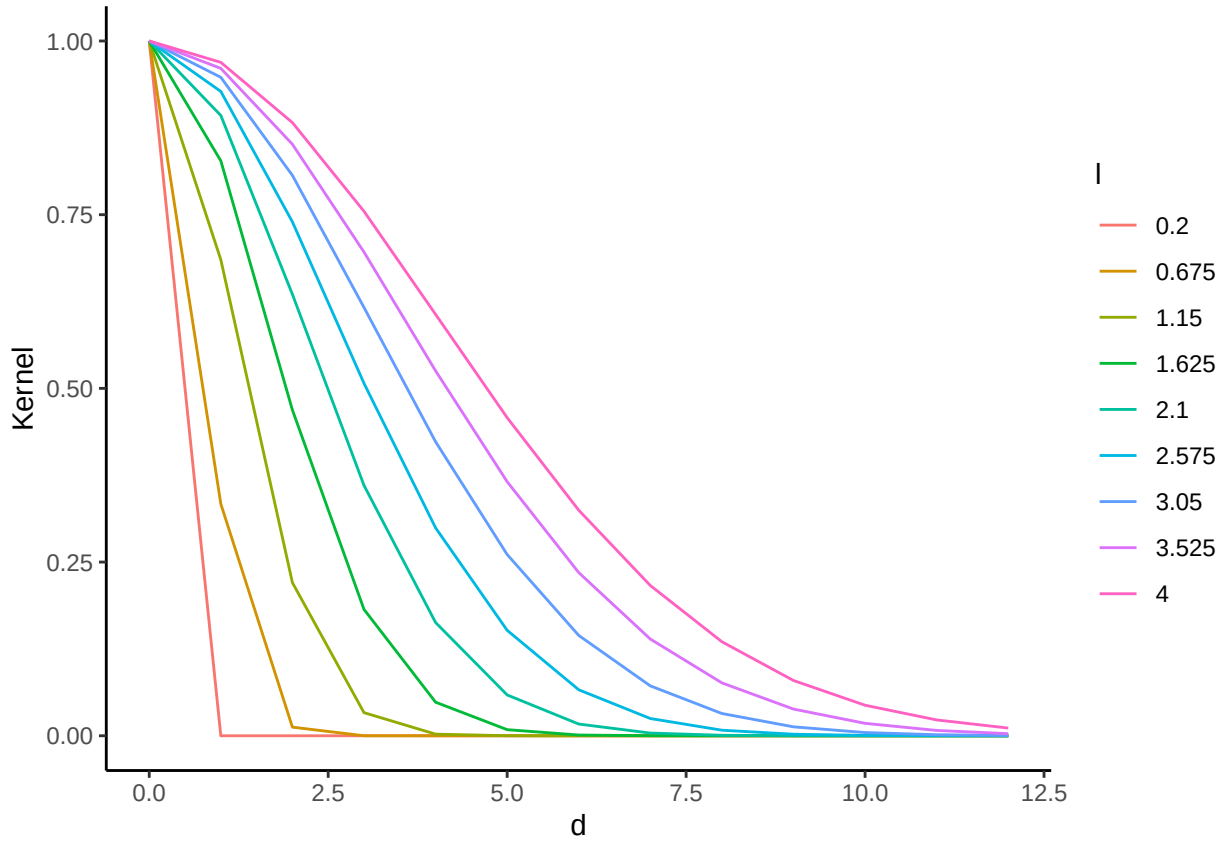
The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

For calculating the time difference during each day, we use a similar attempt to the previous one. However, this time, we check whether the difference is more than 12 hours to use the shorter difference if that is the case.

We pick $l = 1.15$ as a width value because it gives close to zero weight to everything that is more than three hours away.

```
# defining times we want estimations for
times <- c(paste0("0",seq(4,8,by=2),":00:00"),
           paste0(seq(10,24,by=2),":00:00"))
times <- strptime(times, format = "%H:%M:%S")

# change times in dataframe to time object
df$timeD <- strptime(df$time, format = "%H:%M:%S")
diff <- sapply(1:length(times), function(x){
  abs(as.numeric(difftime(times[x], df$timeD), units="hours"))
})
d3 <- ifelse(diff <= 12, diff, 24-diff)
```

```r
# plot different values for l
# x axis from zero up to maximal difference (max(d1))
h3 <- seq(0,max(d3),by=1)

# how many different values do we wanna try and which sequence
m3 <- 9
mySeq3 <- seq(0.2,4,length.out=m3)

# calculate kernel for different values of l
results3 <- lapply(mySeq3, function(l) exp(-h3^2/(2*l^2)))
plot_df3 <- data.frame("Kernel"=unlist(results3),
                       "l"=rep(mySeq3,
                               each=length(h3)),
                       "d"=rep(h3, times=length(m3)))

plot_df3$l <- as.factor(plot_df3$l)
ggplot(plot_df3, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()
```



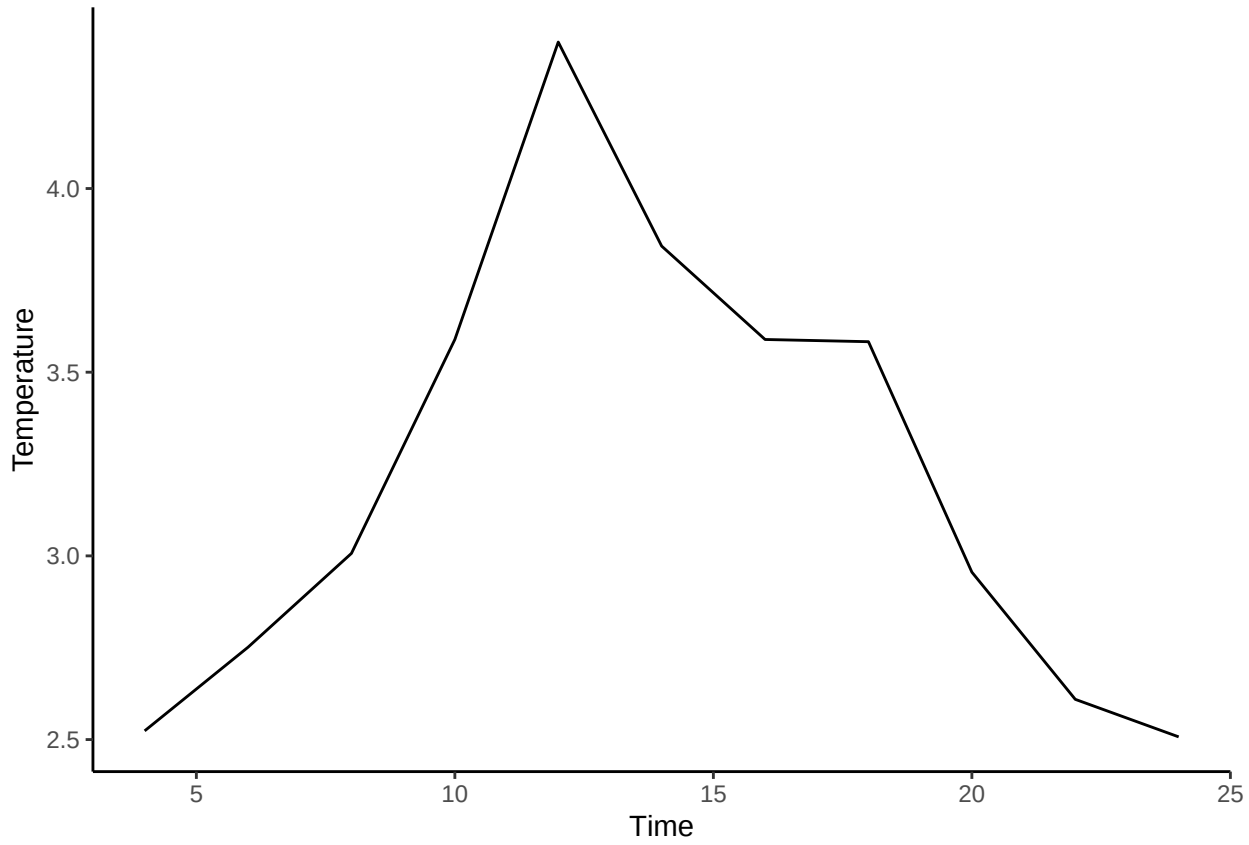

```r
#pick 1.15
#apply to observations
k3 <- exp(-d3^2/(2*1.15^2))
```

Use a kernel that is the sum of three Gaussian kernels.

```
s <- sapply(1:length(times), function(x) sum((k1 + k2 + k3[,x])
                                             *df$air_temperature)/
              sum(k1,k2,k3[,x]))
(rs <- data.frame("Time"=seq(4,24,by=2),"Temperature"=s))
```

```
##    Time Temperature
## 1     4    2.523659
## 2     6    2.751415
## 3     8    3.006832
## 4    10    3.589137
## 5    12    4.398685
## 6    14    3.843309
## 7    16    3.589054
## 8    18    3.583031
## 9    20    2.955756
## 10   22    2.609468
## 11   24    2.507320
```

```
ggplot(rs, aes(y=Temperature,x=Time)) +
  geom_line() +
  theme_classic()
```

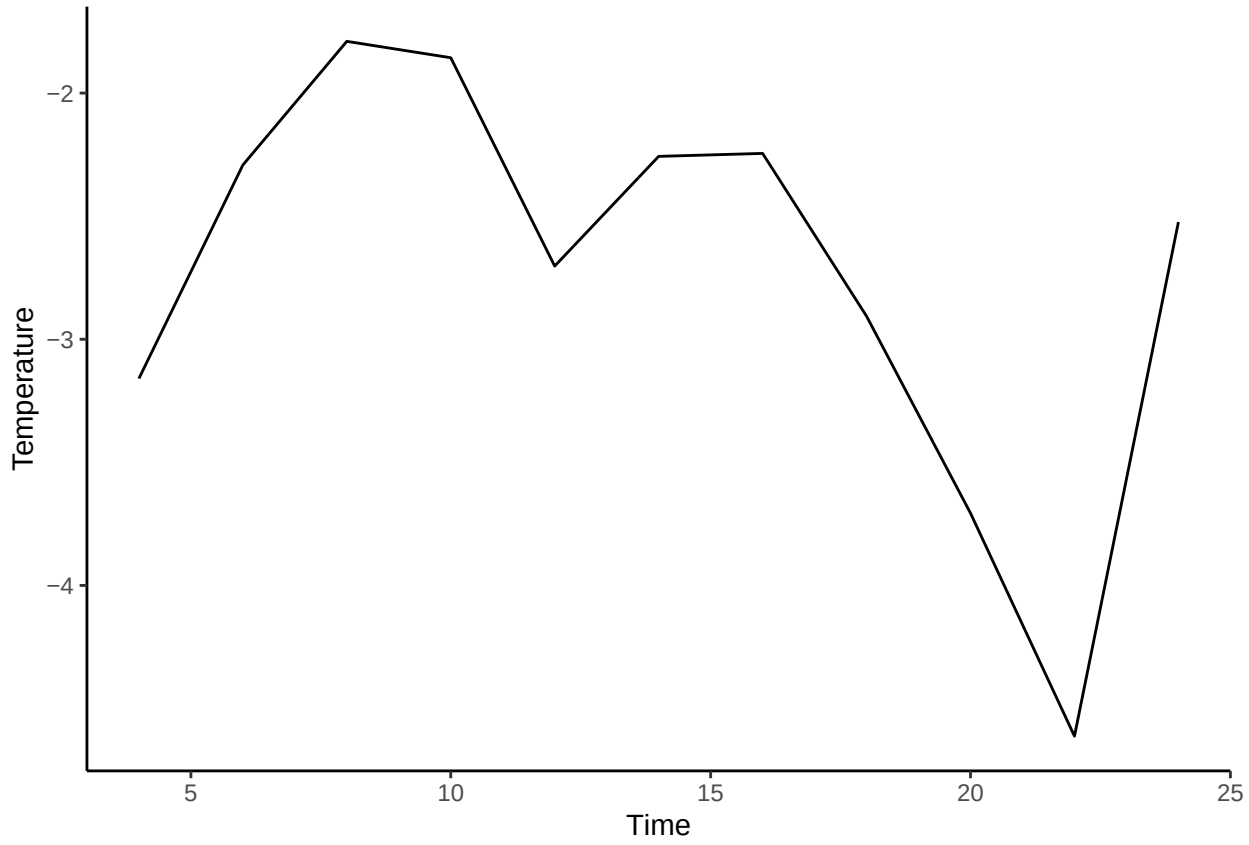


Finally, repeat the exercise above by combining the three kernels into one by multiplying them, instead of summing them up.

```
sM <- sapply(1:length(times), function(x) sum((k1 * k2 * k3[,x])
                                              *df$air_temperature)/
               sum(k1 * k2 * k3[,x]))

(rsM <- data.frame("Time"=seq(4,24,by=2),"Temperature"=sM))
```

```
##    Time Temperature
## 1     4   -3.159492
## 2     6   -2.292204
## 3     8   -1.789908
## 4    10   -1.856450
## 5    12   -2.702746
## 6    14   -2.257094
## 7    16   -2.245127
## 8    18   -2.906534
## 9    20   -3.706508
## 10   22   -4.612227
## 11   24   -2.523919
```

```
ggplot(rsM, aes(y=Temperature,x=Time)) +
  geom_line() +
  theme_classic()
```



Compare the results obtained in both cases and elaborate on why they may differ.

The results for both kernels differ noticeably in their sign as the prediction using multiplication only predicts negative values for the whole day. Of course, this could be right in January as well, so just looking at these values, we can not decide which prediction is more reasonable. However, we picked a date that has a good comparison value and considering this, and we can see that the summation works better in this case:

```
oldDf[which(oldDf$station_name=="Holma"&oldDf$date=="1990-01-29"),]
```

```
##       station_number station_name measurement_height latitude longitude
## 16770          86200        Holma                  2  58.3339   16.8157
##             readings_from         readings_to elevation       date     time
## 16770 1962-05-01 00:00:00 1996-11-30 23:59:59         5 1990-01-29 12:00:00
##       air_temperature quality
## 16770               5       G
```

The location and date were picked to have a measurement during that day close by, which was obviously not used in the prediction but can be used for comparison now. The measured temperature at Holma station was five degrees at lunchtime, and our prediction shows 4.4 degrees at lunchtime which is quite close to the actual value. Nevertheless, it has to be kept in mind that we base this conclusion on only one observation now, which can be seen quite sceptical. More comparison would be necessary for an actual validation of our model, and a real tuning of the different width values would be more sufficient.

We make an additional comparison with another day of the year and then another location as well to see if our results are still reasonable as well:

```
date <- as.Date("1987-07-24")
df <- oldDf[-which(difftime(date, oldDf$date) <= 0),]

#place
d1 <- sapply(1:nrow(df), function(x) distHaversine(tyris,
                                                   c(df[x,]$longitude,
                                                     df[x,]$latitude)))
k1 <- exp(-d1^2/(2*100000^2))

#day
diff <- as.numeric(difftime(date, df$date)) %% 365.25
d2 <- ifelse(diff <= 182.625, diff, 365.25-diff)
k2 <- exp(-d2^2/(2*11^2))

#time
df$timeD <- strptime(df$time, format = "%H:%M:%S")
diff <- sapply(1:length(times), function(x){
  abs(as.numeric(difftime(times[x], df$timeD), units="hours"))
})
d3 <- ifelse(diff <= 12, diff, 24-diff)
k3 <- exp(-d3^2/(2*1.15^2))
```

```
#summation kernel
s <- sapply(1:length(times), function(x) sum((k1 + k2 + k3[,x])
                                              *df$air_temperature)/
              sum(k1,k2,k3[,x]))
data.frame("Time"=seq(4,24,by=2),"Temperature"=s)
```

```
##    Time Temperature
## 1     4    7.268561
## 2     6    6.435890
## 3     8    7.631886
## 4    10    8.169273
## 5    12    7.998995
## 6    14    8.416319
## 7    16    8.202768
## 8    18    7.277033
## 9    20    7.845308
## 10   22    7.995713
## 11   24    7.542507
```

```
#multiplication kernel
sM <- sapply(1:length(times), function(x) sum((k1 * k2 * k3[,x])
                                              *df$air_temperature)/
              sum(k1 * k2 * k3[,x]))

data.frame("Time"=seq(4,24,by=2),"Temperature"=sM)
```

```
##    Time Temperature
## 1     4    13.45444
## 2     6    14.98427
## 3     8    16.92612
## 4    10    18.89165
## 5    12    19.43370
## 6    14    19.37360
## 7    16    18.73687
## 8    18    17.73583
## 9    20    16.23010
## 10   22    14.86001
## 11   24    14.15185
```

Now, we picked a day in summer and in this case, the multiplication of the different kernels is a lot more successful as the values are a lot more reasonable for temperatures in July. In general, it makes sense that we get nicer values for the multiplication as it is a lot stricter in punishing values that are far away (in all terms of distances), as multiplying by a number close to zero is a lot stronger than adding a number close to zero to a sum of two other numbers. We are not quite sure why the multiplication was not performing that well for our prediction in winter, but we assume that it has something to do with the temperatures around zero.

```
#place
umea <- c(20.263035,63.825848)
d1 <- sapply(1:nrow(df), function(x) distHaversine(umea,
                                                   c(df[x,]$longitude,
                                                     df[x,]$latitude)))
```

```r
k1 <- exp(-d1^2/(2*100000^2))

#the rest we use from before
s <- sapply(1:length(times), function(x) sum((k1 + k2 + k3[,x])
                                             *df$air_temperature)/
              sum(k1,k2,k3[,x]))
data.frame("Time"=seq(4,24,by=2),"Temperature"=s)
```

```
##    Time Temperature
## 1     4    6.595765
## 2     6    5.745273
## 3     8    7.105636
## 4    10    7.829137
## 5    12    7.708718
## 6    14    8.163008
## 7    16    7.873990
## 8    18    6.794502
## 9    20    7.357716
## 10   22    7.503634
## 11   24    6.924940
```

```r
sM <- sapply(1:length(times), function(x) sum((k1 * k2 * k3[,x])
                                             *df$air_temperature)/
              sum(k1 * k2 * k3[,x]))

data.frame("Time"=seq(4,24,by=2),"Temperature"=sM)
```

```
##    Time Temperature
## 1     4    12.79479
## 2     6    13.70910
## 3     8    15.19957
## 4    10    18.20760
## 5    12    17.61572
## 6    14    16.94201
## 7    16    16.68685
## 8    18    16.69046
## 9    20    15.23084
## 10   22    12.95040
## 11   24    11.24332
```

Even for another location, it seems to be the same that multiplication is working quite well while summation is not. However, it could also be quite cold up north in Umea...

Assignment 2: Support Vector Machines

```
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58]) # Scale data except for response (type)
tr <- spam[1:3000, ] # Training data
va <- spam[3001:3800, ] # Validation data
trva <- spam[1:3800, ] # Training and validation data
te <- spam[3801:4601, ] # Testing data

by <- 0.3
# seq(0.3, 5, by = 0.3) -> 16 models with different values for the cost of
# constraints violation.
# C is the Its the regularization term in the Lagrange formulation
err_va <- NULL # Error on validation data
for(i in seq(by,5,by)){
  # Runs 16 different models with different C
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",
                 kpar=list(sigma=0.05),C=i,scaled=FALSE)

  mailtype <- predict(filter,va[,-58]) # Predict on validation
  t <- table(mailtype,va[,58]) # confusion matrix
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t)) # error on validation data
}

# Some different models with C=3.9 (most optimal based on validation error)

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

```
## [1] 0.0675
```

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

```
## [1] 0.08489388
```

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

```
## [1] 0.082397
```

```
### Training on all the data
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",
                kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
## Predict testing
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3 # Error on testing
```

```
## [1] 0.02122347
```

Questions

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

The filter1 is the most proper model as the one that should be returned to the user. This model is trained using the training data and has had hyperparameters (C) tuned by using the validation data. It therefore has an unbiased estimate of the error since the testing data is unknown to the model.

Returning the other models all bring some sort of issues. Filter0 is computing error on the validation data that was used to tune hyperparameter C and is therefore not testing on unbiased new data. We cannot then know how well the model generalize to new data. Filter2 uses both the training data and the validation data to fit the model and then tests on the testing data. Using the validation data first to tune the hyperparameter and then to retrain the model using both the training and validation data may be overfitting to the validation data and will in theory result in a model that is worse at generalizing to new data. Filter3 uses all of the data (including the testing data) to fit the model and this creates reliability issues in terms of the error rate on testing data as the model becomes overfit to the testing data and we cannot be certain how well it will generalize to new data.

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

The generalization error is `err1` since this is the only error that is truly new and unseen by the model. The *err2* also uses unseen data, but the validation is already used for calculating C and then again for training the model, making it more biased towards the validation data.

```
cat("The generaliztion error is:", round(err1,4))
```

```
## The generaliztion error is: 0.0849
```

3. Implementation of SVM predictions.

```
# 3. Implementation of SVM predictions.

sv<-alphaindex(filter3)[[1]] # indexes of the support vectors
co<-coef(filter3)[[1]] # the linear coefficients for the support vectors
inte<- - b(filter3) # the negative intercept of the linear combination

# Gaussian kernel
Gkernel <- function(x, x_new, sigma = 0.05){
  return(exp(-sigma*sum((x-x_new)^2)))
}

k<-numeric(10) # Vector to hold the predictions
k2 <- matrix(0, ncol = length(sv), nrow = 10) # Matrix to hold the alpha_j * K(X_j, X_new)
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  for(j in 1:length(sv)){ # Using only the support vectors
    # Take alpha_i times the kernel value of each support vector with the new observation
    k2[i,j] <- co[j] * Gkernel(spam[sv[j],-58], spam[i,-58])
  }
  # Sum up all the kernelvalues scaled by alpha and add the intercept
  k[i] <- sum(k2[i,]) + inte
}

cat("Does our solution and predict() return the same function? Answer:",
    all.equal(k, as.numeric(predict(filter3,spam[1:10,-58], type = "decision"))))
```

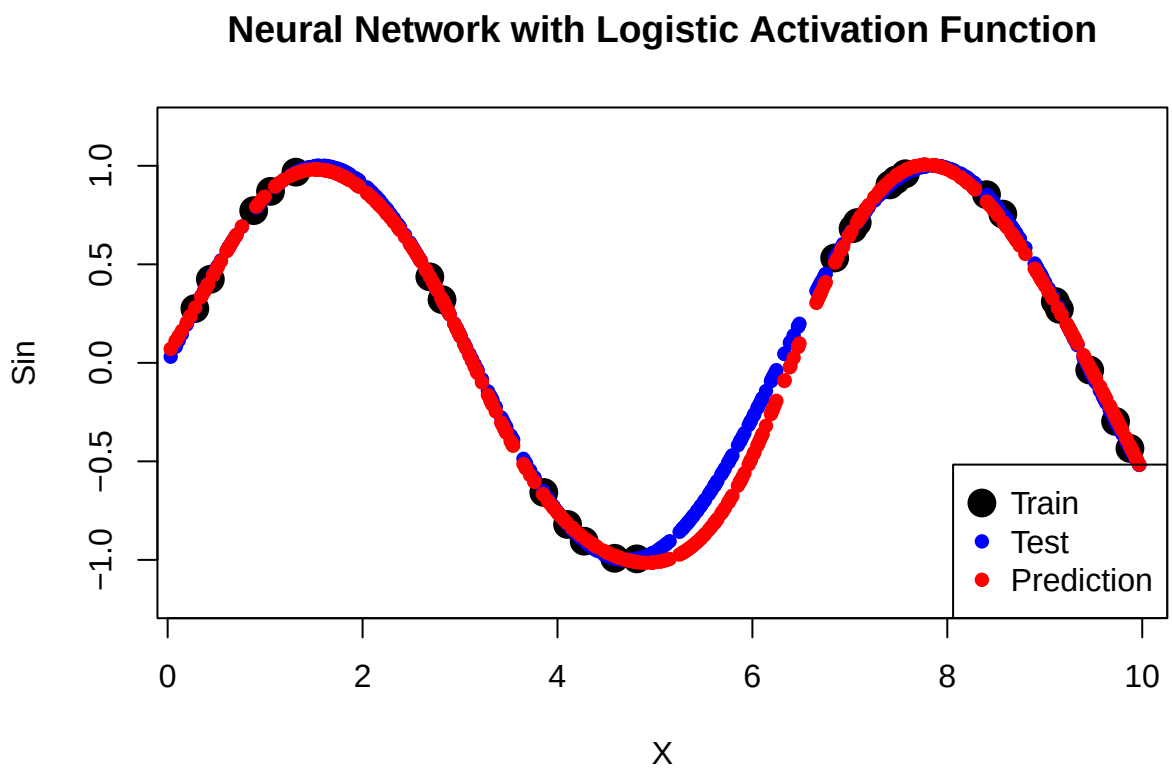
```
## Does our solution and predict() return the same function? Answer: TRUE
```

Assignment 3: Neural Networks

Task 1: Construct Data & Implement Neural Network

```
library(neuralnet)
set.seed(1234567890)
X <- runif(500, 0, 10)
data1 <- data.frame(X, Sin=sin(X))
train <- data1[1:25,] # Training
test <- data1[26:500,] # Test
```

```
nn <- neuralnet(Sin~X, hidden = 10, data = train) # default activation function logistic

pred_test <- predict(nn, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Logistic Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```



Using the logistic function, $f(x) = \frac{1}{1+e^x}$, as an activation function (default) the predictions of the learned model on the test data are relatively good. The only difference between the predicted points (red) and the

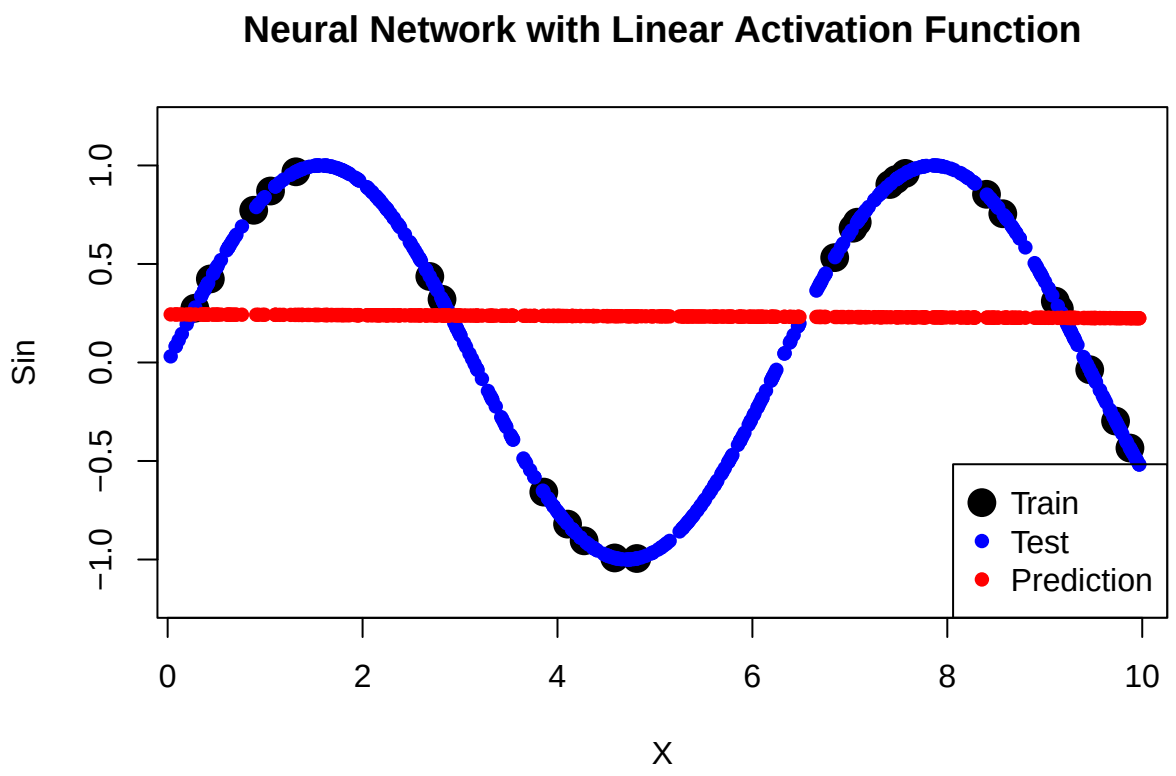
actual point (black for train and blue for test) is when X has values between 5 and 6.5. This is due the fact that there are no training points in that interval.

The logistic activation function is the most commonly used for training neural network models where we have to predict the output as a probability. The sigmoid function is the most suitable choice because its range is between 0 and 1, the range of any probability. Moreover, the function is differentiable, and as a result, it provides a smooth gradient.

Task 2: Repeat question (1) with custom activation functions.

Linear activation function

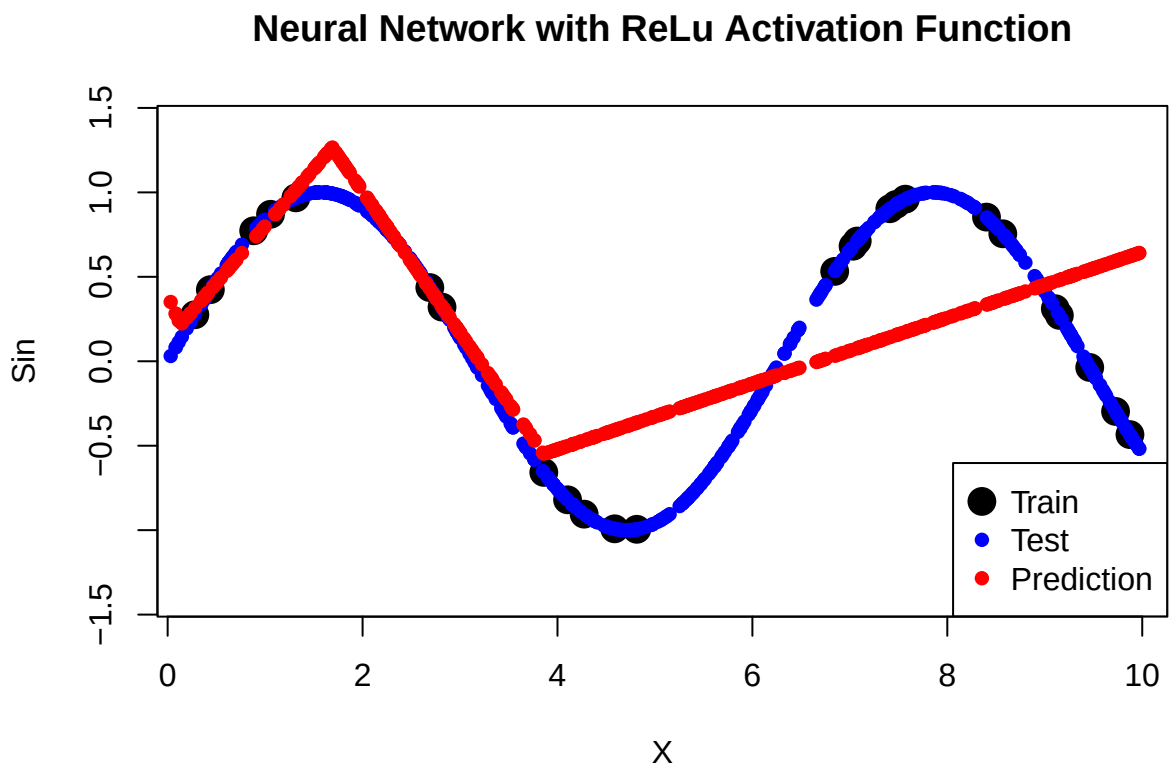
```
h1 <- function(x) {x}
nn1 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h1)

pred_test_1 <- predict(nn1, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Linear Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_1, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```



The graph illustrates that the predictions of the learned neural networks on the test data are a straight line (red points), compared to the train (black points) and test (blue points) values, which correctly demonstrate the plot of sine. That happens because it is not possible to use backpropagation as the derivative of the linear function, $f(x) = x$, is a constant and has no relation to the input x. Thus, all neural network layers will collapse into one if a linear activation function is used. No matter the number of layers, the linear activation function turns the neural network into one layer.

ReLU activation function

```
h2 <- function(x){ifelse(x>=0,x,0)}
# h2 <- function(x){max(0,x)} # does not work.

nn2 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h2)

pred_test_2 <- predict(nn2, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.4,1.4),
     main = "Neural Network with ReLu Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_2, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```

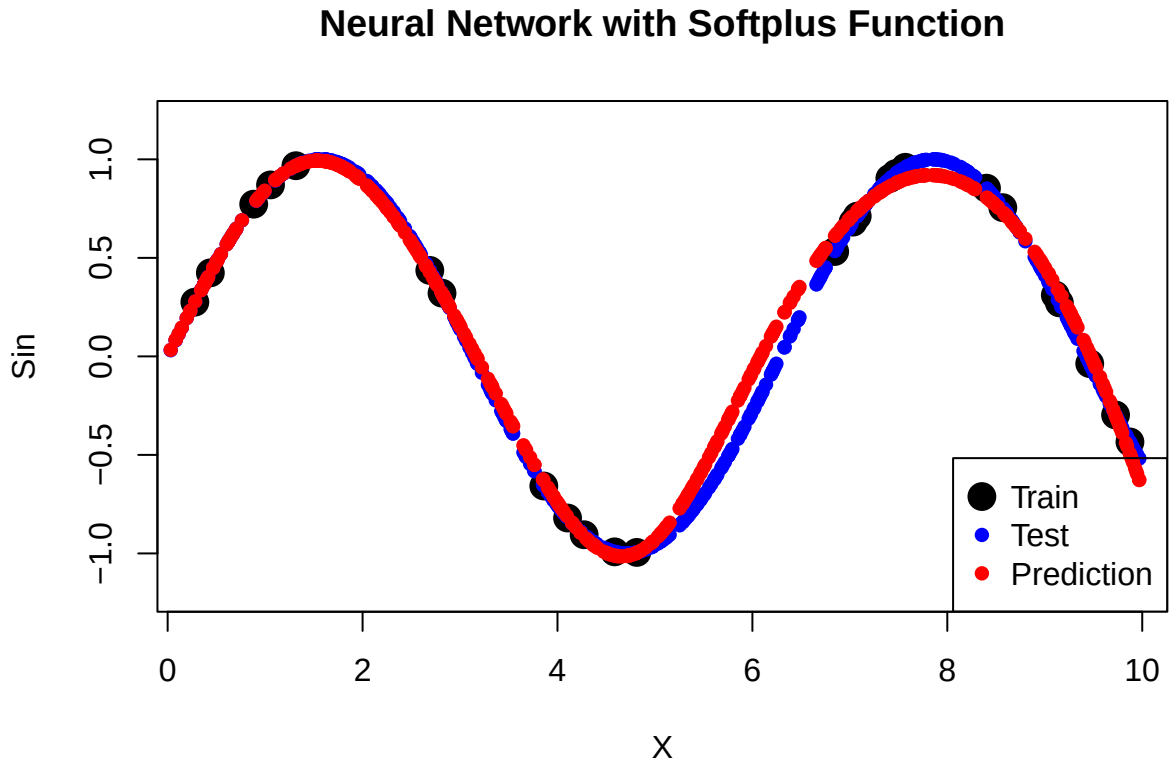


Using the ReLU function as an activation function the predictions of the learned model on the test data do not provide good results. The only predictions that illustrate the sine plot correctly are when X has values between 0.5 and 1.5 and from 2.2 to 3.5. The only problem with this activation function is that when x is smaller than zero, the function returns zero; thus, during the backpropagation process, the weights and biases for some neurons are not updated, which could create dead neurons that never get activated.

Moreover, defining the ReLU function in R as $f(x) = max(0, x)$, does not work and it returns an error because it is not differentiable when x equals zero. The most common solution is to set x equal 1, where the derivative of 1 is zero.

Softplus activation function

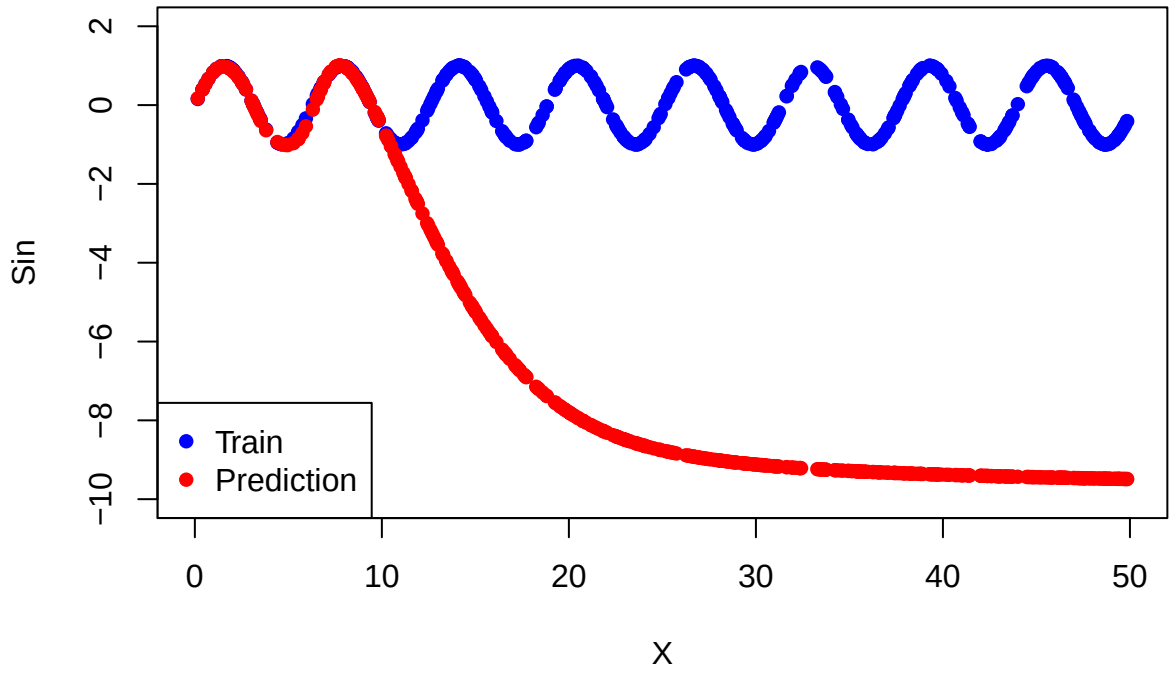
```
h3 <- function(x) log(1 + exp(x))

nn3 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h3)

pred_test_3 <- predict(nn3, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Softplus Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_3, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
```



The above plot shows that the predictions of the learned neural network, which uses the softplus function $f(x) = \log(1 + e^x)$ as an activation function, on the test data provide the best results compared to the previous two examples. The softplus activation function is a smooth approximation of the ReLU activation function.

Task 3: New data and predictions using the initial NN model.

```
set.seed(1234567890)
X <- runif(500, 0, 50)
data2 <- data.frame(X, Sin=sin(X))

pred <- predict(nn, data2)

plot(data2, cex=1, col = "blue", pch = 16, xlim = c(0,50), ylim = c(-10,2))
points(data2[,1],pred, col="red", cex=1, pch = 16)
legend("bottomleft", legend = c("Train", "Prediction"),
       col = c("blue", "red"), pch = 16, pt.cex = c(1,1))
```



The predictions that the trained model returns illustrate sine correctly when X has values between 0 and 10, which makes sense because the initialized neural network model is trained with values between 0 and 10.

Task 4: Role of the weights.

Our model has:

$$\boldsymbol{W^{(1)}} = \begin{bmatrix} -1.84849659 \\ -1.10815997 \\ 0.26168253 \\ -2.29815442 \\ -0.05618884 \\ -1.96773214 \\ -2.51025218 \\ -1.10418652 \\ -0.76154882 \\ 1.75588529 \end{bmatrix}, \qquad \boldsymbol{b^{(1)}} = \begin{bmatrix} 3.9654060 \\ -1.2313394 \\ -2.7255077 \\ 7.3872163 \\ -0.4392372 \\ 0.8143266 \\ 16.9497283 \\ -0.8458926 \\ 2.4703284 \\ -11.6184909 \end{bmatrix}$$

and

$$\boldsymbol{W^{(2)}} = \begin{bmatrix} 0.6261725 \\ -1.3652234 \\ -15.2546738 \\ 1.4724743 \\ 4.0194898 \\ -2.3705710 \\ 1.5225230 \\ 1.3549643 \\ -1.9815383 \\ 6.4412888 \end{bmatrix}, \qquad \boldsymbol{b^{(2)}} = \begin{bmatrix} -0.8272835 \end{bmatrix}$$

As can be seen in $\boldsymbol{W^{(1)}}$, there are only two weights that are positive: W_3 and W_{10}. If we calculate $\boldsymbol{W^{(1)}X} + \boldsymbol{b^{(1)}}$, where $\boldsymbol{X}$ is simply a large scalar value then the resulting matrix will have large negative values in the rows where the weight is negative and large positive values where the weight is positive. Typically the weights makes the input even larger. There are two exceptions, W_5 and W_2 of $\boldsymbol{W^{(1)}}$ which have values in the interval $(-1, 0)$ meaning that the input gets scaled down. If the input is very large this wont however make any difference for the calculation but has an effect before the prediction converges. Now consider when the input is large, say 100, then applying the sigmoid function to $\boldsymbol{W^{(1)}X} + \boldsymbol{b^{(1)}}$. When the values in $\boldsymbol{W^{(1)}X} + \boldsymbol{b^{(1)}}$ are large negative numbers then the sigmoid function will transform them to be closer and closer to zero as the input becomes larger and larger. When the values are large positive number, applying the sigmoid function will make them closer and closer to one. As the input increases, $\boldsymbol{q}$ will thus approach:

$$\boldsymbol{q} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Then doing next step in the neural network calculations:

$$\hat{y} = \boldsymbol{W^{(2)}}\boldsymbol{q} + \boldsymbol{b^{(2)}}$$

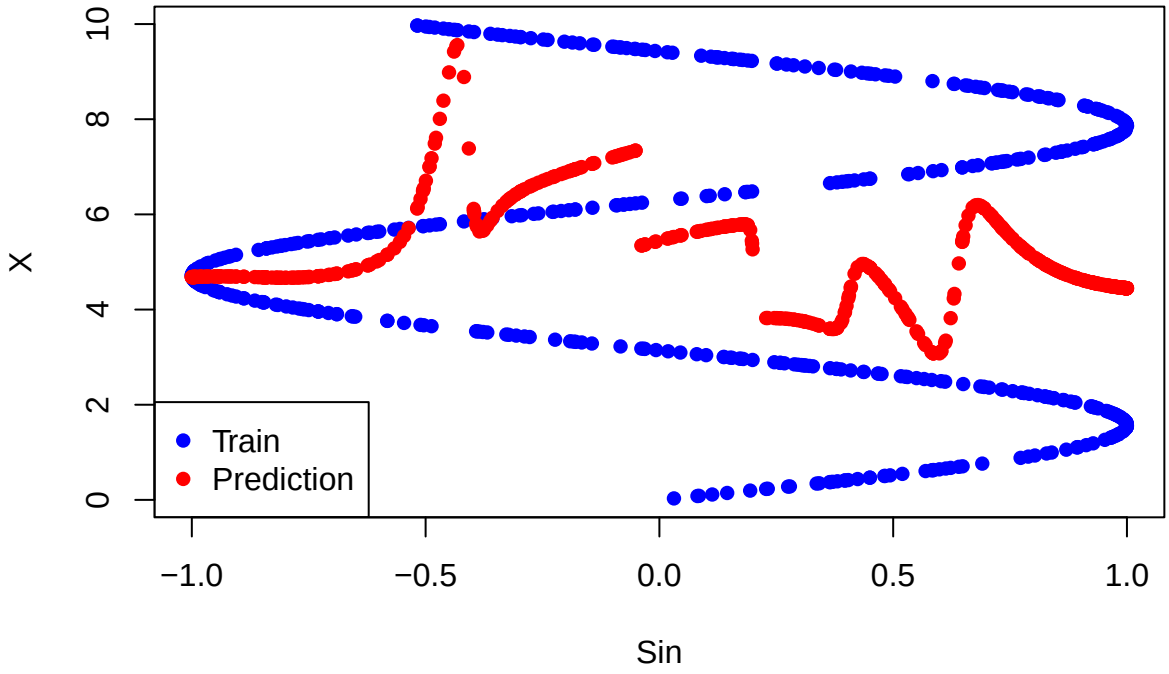
will be equivalent to simply calculating the sum $W_3^{(2)} + W_{10}^{(2)} + \boldsymbol{b^{(2)}} = -15.2546738 + 6.4412888 - 0.8272835 = -9.63129$.

```
# Sigmoid function
sigmoid <- function(x){
  return(1 / (1 + exp(-x)))
}
# Define weight matrix for layer 1
W1 <- nn$weights[[1]][[1]][2,]
# Define offset vector for layer 1
b1 <- nn$weights[[1]][[1]][1,]
# Get q: Test with input 100
q <- sigmoid(W1 * 100 + b1)
#
b2 <- nn$weights[[1]][[2]][1,]
W2 <- nn$weights[[1]][[2]][2:11,]
cat("The value will converge to:", W2%*%q + b2, "as the input increases")
```

```
## The value will converge to: -9.63129 as the input increases
```

Task 5: New NN model to predict x from sin(x)

```
nn5 <- neuralnet(X~Sin, hidden = 10, data = data1, threshold = 0.1)

pred_data <- predict(nn5, data1)

plot(data1$Sin, data1$X, col="blue", cex=1, pch = 16, xlab = "Sin", ylab = "X")
points(data1[,2],pred_data, col="red", cex=1, pch = 16)
legend("bottomleft", legend = c("Train", "Prediction"),
       col = c("blue", "red"), pch = 16, pt.cex = c(1,1))
```



The predicted results do not present the actual values; the reason is that the neural network model is confused because sine returns the same results for several x values.

Statement of Contribution

We first worked on all assignments ourselves, and everyone finished all of them. Lisa was mainly concentrating on assignment 1 and wrote down the interpretations there. Theodor focused most on assignment 2 and wrote the interpretation for these tasks while Christoforos was doing the same for assignment 3. After everyone finished, we discussed our results and adjusted a few things.

Code

```
knitr::opts_chunk$set(echo = TRUE)
library(geosphere)
library(ggplot2)
library(kernlab)
set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")

#all data together
df <- merge(stations,temps,by="station_number")
# date we are choosing (can make a comparison to Holma station
# later and we need to define the date now for filtering data)
date <- as.Date("1990-01-29")

# making date in the dataset a date.object to use difftime to filter out all
# data from this day and after
df$date <- as.Date(df$date)
oldDf <- df #later comparison
df <- df[-which(difftime(date, df$date) <= 0),]
# place we are going to pick is Tyrislöt (points from google maps)
tyris <- c(16.89461,58.32633)

# calculate distance
d1 <- sapply(1:nrow(df), function(x) distHaversine(tyris,
                                                   c(df[x,]$longitude,
                                                     df[x,]$latitude)))

# plot different values for l
# x axis from zero up to maximal difference (max(d1))
h1 <- seq(0,max(d1),by=1000)

# how many different values do we wanna try and which sequence
m1 <- 9
mySeq1 <- seq(50000,130000,length.out=m1)

# calculate kernel for different values of l
results1 <- lapply(mySeq1, function(l) exp(-h1^2/(2*l^2)))
plot_df1 <- data.frame("Kernel"=unlist(results1),
                       "l"=rep(mySeq1,
                               each=length(h1)),
                       "d"=rep(h1, times=length(m1)))
plot_df1$l <- as.factor(plot_df1$l)
```

```r
ggplot(plot_df1, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()

# pick 100000
# apply to observations
k1 <- exp(-d1^2/(2*100000^2))
# date we are choosing is the 29th of January 1990

# calculate difference
diff <- as.numeric(difftime(date, df$date)) %% 365.25
d2 <- ifelse(diff <= 182.625, diff, 365.25-diff)

# plot different values for l
# x axis from zero up to maximal difference (max(d2))
h2 <- seq(0,max(d2),by=1)

# how many different values do we wanna try and which sequence
m2 <- 9
mySeq2 <- seq(2,74,length.out=m2)

# calculate kernel for different values of l
results2 <- lapply(mySeq2, function(l) exp(-h2^2/(2*l^2)))
plot_df2 <- data.frame("Kernel"=unlist(results2),
                       "l"=rep(mySeq2,
                               each=length(h2)),
                       "d"=rep(h2, times=length(m2)))

plot_df2$l <- as.factor(plot_df2$l)
ggplot(plot_df2, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()

# pick 11
# apply to observations
k2 <- exp(-d2^2/(2*11^2))
# defining times we want estimations for
times <- c(paste0("0",seq(4,8,by=2),":00:00"),
           paste0(seq(10,24,by=2),":00:00"))
times <- strptime(times, format = "%H:%M:%S")

# change times in dataframe to time object
df$timeD <- strptime(df$time, format = "%H:%M:%S")
diff <- sapply(1:length(times), function(x){
  abs(as.numeric(difftime(times[x], df$timeD), units="hours"))
})
d3 <- ifelse(diff <= 12, diff, 24-diff)

# plot different values for l
# x axis from zero up to maximal difference (max(d1))
h3 <- seq(0,max(d3),by=1)

# how many different values do we wanna try and which sequence
```

```
m3 <- 9
mySeq3 <- seq(0.2,4,length.out=m3)

# calculate kernel for different values of l
results3 <- lapply(mySeq3, function(l) exp(-h3^2/(2*l^2)))
plot_df3 <- data.frame("Kernel"=unlist(results3),
                       "l"=rep(mySeq3,
                               each=length(h3)),
                       "d"=rep(h3, times=length(m3)))

plot_df3$l <- as.factor(plot_df3$l)
ggplot(plot_df3, aes(x=d,y=Kernel,col=l)) +
  geom_line() +
  theme_classic()

#pick 1.15
#apply to observations
k3 <- exp(-d3^2/(2*1.15^2))
s <- sapply(1:length(times), function(x) sum((k1 + k2 + k3[,x])
                                             *df$air_temperature)/
              sum(k1,k2,k3[,x]))
(rs <- data.frame("Time"=seq(4,24,by=2),"Temperature"=s))

ggplot(rs, aes(y=Temperature,x=Time)) +
  geom_line() +
  theme_classic()
sM <- sapply(1:length(times), function(x) sum((k1 * k2 * k3[,x])
                                             *df$air_temperature)/
              sum(k1 * k2 * k3[,x]))

(rsM <- data.frame("Time"=seq(4,24,by=2),"Temperature"=sM))

ggplot(rsM, aes(y=Temperature,x=Time)) +
  geom_line() +
  theme_classic()
oldDf[which(oldDf$station_name=="Holma"&oldDf$date=="1990-01-29"),]
date <- as.Date("1987-07-24")
df <- oldDf[-which(difftime(date, oldDf$date) <= 0),]

#place
d1 <- sapply(1:nrow(df), function(x) distHaversine(tyris,
                                                   c(df[x,]$longitude,
                                                     df[x,]$latitude)))
k1 <- exp(-d1^2/(2*100000^2))

#day
diff <- as.numeric(difftime(date, df$date)) %% 365.25
d2 <- ifelse(diff <= 182.625, diff, 365.25-diff)
k2 <- exp(-d2^2/(2*11^2))

#time
df$timeD <- strptime(df$time, format = "%H:%M:%S")
diff <- sapply(1:length(times), function(x){
```

```
  abs(as.numeric(difftime(times[x], df$timeD), units="hours"))
})
d3 <- ifelse(diff <= 12, diff, 24-diff)
k3 <- exp(-d3^2/(2*1.15^2))
#summation kernel
s <- sapply(1:length(times), function(x) sum((k1 + k2 + k3[,x])
                                             *df$air_temperature)/
              sum(k1,k2,k3[,x]))
data.frame("Time"=seq(4,24,by=2),"Temperature"=s)

#multiplication kernel
sM <- sapply(1:length(times), function(x) sum((k1 * k2 * k3[,x])
                                             *df$air_temperature)/
              sum(k1 * k2 * k3[,x]))

data.frame("Time"=seq(4,24,by=2),"Temperature"=sM)
#place
umea <- c(20.263035,63.825848)
d1 <- sapply(1:nrow(df), function(x) distHaversine(umea,
                                                   c(df[x,]$longitude,
                                                     df[x,]$latitude)))
k1 <- exp(-d1^2/(2*100000^2))

#the rest we use from before
s <- sapply(1:length(times), function(x) sum((k1 + k2 + k3[,x])
                                             *df$air_temperature)/
              sum(k1,k2,k3[,x]))
data.frame("Time"=seq(4,24,by=2),"Temperature"=s)

sM <- sapply(1:length(times), function(x) sum((k1 * k2 * k3[,x])
                                             *df$air_temperature)/
              sum(k1 * k2 * k3[,x]))

data.frame("Time"=seq(4,24,by=2),"Temperature"=sM)

set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58]) # Scale data except for response (type)
tr <- spam[1:3000, ] # Training data
va <- spam[3001:3800, ] # Validation data
trva <- spam[1:3800, ] # Training and validation data
te <- spam[3801:4601, ] # Testing data

by <- 0.3
# seq(0.3, 5, by = 0.3) -> 16 models with different values for the cost of
# constraints violation.
# C is the Its the regularization term in the Lagrange formulation
err_va <- NULL # Error on validation data
for(i in seq(by,5,by)){
  # Runs 16 different models with different C
```

```
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",
                 kpar=list(sigma=0.05),C=i,scaled=FALSE)

  mailtype <- predict(filter,va[,-58]) # Predict on validation
  t <- table(mailtype,va[,58]) # confusion matrix
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t)) # error on validation data
}

# Some different models with C=3.9 (most optimal based on validation error)

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

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

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

### Training on all the data
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",
                kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
## Predict testing
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3 # Error on testing
cat("The generaliztion error is:", round(err1,4))
# 3. Implementation of SVM predictions.

sv<-alphaindex(filter3)[[1]] # indexes of the support vectors
co<-coef(filter3)[[1]] # the linear coefficients for the support vectors
inte<- - b(filter3) # the negative intercept of the linear combination

# Gaussian kernel
```

```r
Gkernel <- function(x, x_new, sigma = 0.05){
  return(exp(-sigma*sum((x-x_new)^2)))
}

k<-numeric(10) # Vector to hold the predictions
k2 <- matrix(0, ncol = length(sv), nrow = 10) # Matrix to hold the alpha_j * K(X_j, X_new)
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  for(j in 1:length(sv)){ # Using only the support vectors
    # Take alpha_i times the kernel value of each support vector with the new observation
    k2[i,j] <- co[j] * Gkernel(spam[sv[j],-58], spam[i,-58])
  }
  # Sum up all the kernelvalues scaled by alpha and add the intercept
  k[i] <- sum(k2[i,]) + inte
}

cat("Does our solution and predict() return the same function? Answer:",
    all.equal(k, as.numeric(predict(filter3,spam[1:10,-58], type = "decision"))))
library(neuralnet)
set.seed(1234567890)
X <- runif(500, 0, 10)
data1 <- data.frame(X, Sin=sin(X))
train <- data1[1:25,] # Training
test <- data1[26:500,] # Test
nn <- neuralnet(Sin~X, hidden = 10, data = train) # default activation function logistic

pred_test <- predict(nn, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Logistic Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
h1 <- function(x) {x}
nn1 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h1)

pred_test_1 <- predict(nn1, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Linear Activation Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_1, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
h2 <- function(x){ifelse(x>=0,x,0)}
# h2 <- function(x){max(0,x)} # does not work.

nn2 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h2)

pred_test_2 <- predict(nn2, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.4,1.4),
     main = "Neural Network with ReLu Activation Function")
```

```
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_2, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
h3 <- function(x) log(1 + exp(x))

nn3 <- neuralnet(Sin~X, hidden = 10, data = train, act.fct = h3)

pred_test_3 <- predict(nn3, test)

plot(train, cex=2, col = "black", pch = 16, ylim = c(-1.2,1.2),
     main = "Neural Network with Softplus Function")
points(test, col = "blue", cex=1, pch = 16)
points(test[,1],pred_test_3, col="red", cex=1, pch = 16)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
       col = c("black", "blue", "red"), pch = 16, pt.cex = c(2,1,1))
set.seed(1234567890)
X <- runif(500, 0, 50)
data2 <- data.frame(X, Sin=sin(X))

pred <- predict(nn, data2)

plot(data2, cex=1, col = "blue", pch = 16, xlim = c(0,50), ylim = c(-10,2))
points(data2[,1],pred, col="red", cex=1, pch = 16)
legend("bottomleft", legend = c("Train", "Prediction"),
       col = c("blue", "red"), pch = 16, pt.cex = c(1,1))
plot(nn)
# Sigmoid function
sigmoid <- function(x){
  return(1 / (1 + exp(-x)))
}
# Define weight matrix for layer 1
W1 <- nn$weights[[1]][[1]][2,]
# Define offset vector for layer 1
b1 <- nn$weights[[1]][[1]][1,]
# Get q: Test with input 100
q <- sigmoid(W1 * 100 + b1)
#
b2 <- nn$weights[[1]][[2]][1,]
W2 <- nn$weights[[1]][[2]][2:11,]
cat("The value will converge to:", W2%*%q + b2, "as the input increases")
nn5 <- neuralnet(X~Sin, hidden = 10, data = data1, threshold = 0.1)

pred_data <- predict(nn5, data1)

plot(data1$Sin, data1$X, col="blue", cex=1, pch = 16, xlab = "Sin", ylab = "X")
points(data1[,2],pred_data, col="red", cex=1, pch = 16)
legend("bottomleft", legend = c("Train", "Prediction"),
       col = c("blue", "red"), pch = 16, pt.cex = c(1,1))
```