Machine Learning Lab 3

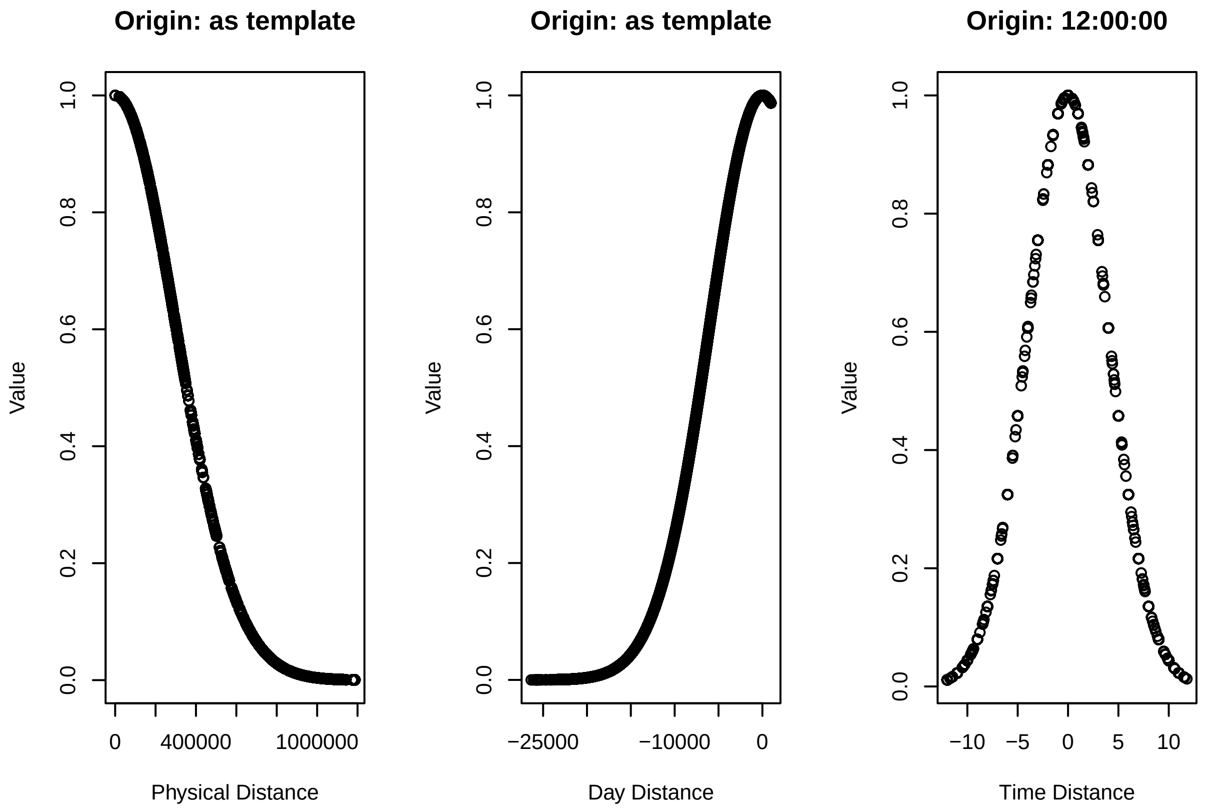
Xuan Wang, Priyarani Patil, Lepeng Zhang

2023-12-17

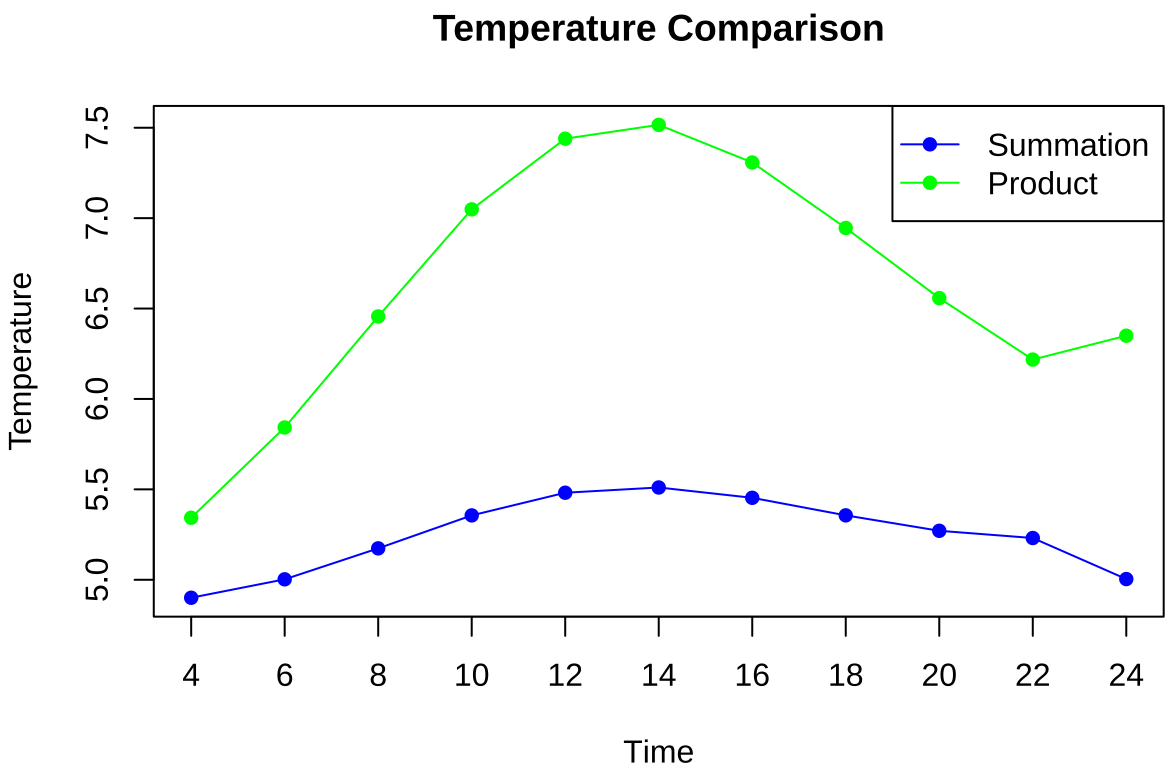
Statement of Contribution

The group report was made based on the discussion after all of us had finished all three assignments. Assignment 1 was mainly contributed by Lepeng Zhang. Assignment 2 was mainly contributed by Priyarani Patil. Assignment 3 was mainly contributed by Xuan Wang.

Assignment 1. KERNEL METHODS



```
## Temp (the kernel is the sum of three Gaussian kernels):

##  [1] 4.900597 5.002446 5.173720 5.356002 5.481321 5.510354 5.453255 5.356227
##  [9] 5.270820 5.230959 5.003899

## Temp (the kernel is the product of three Gaussian kernels):

##  [1] 5.342949 5.842327 6.456291 7.048627 7.439454 7.515983 7.307953 6.945617
##  [9] 6.557870 6.217806 6.349853
```



The comparison graph shows that the temp values computed by the summation of kernel values stay in a relatively narrow range over the interested time period compared with their counterparts which are computed by the product of kernel values.
By checking the ranges of kernel values obtained by summation and product, it can be found that the ratio of maximum and minimum of summation kernel values is 2.94 (2.997843/1.018752), while this value of product kernel values is 5292.28 (0.9978443/0.0001885472) (the specific value can be changed with different interested hour and h values). This indicates that for product situation, the sample point which is close to the interested point contributes significantly to the final result and the remote sample point contributes very little. The contribution gap is huge. While for summation situation, this contribution gap is relatively small. No matter a sample point is close or far away from the interested point, it contributes a certain extent to the final result. Based on this, we believe that the product kernel value is preferred in this task since the weather can change a lot even within a short period or a small region. In this sense, the weather data obtained far away from the interested place and time has no much value for forecast while the data obtained closely contributes significantly.

Assignment 2. SUPPORT VECTOR MACHINES

(1)

All these four filters are set the parameter **C** to the value that leads to the minimal validation error. While, this batch of validation errors are obtained based on **tr** training set and **va** validation set. So to achieve the minimal validation error for the model, the filter should be trained with **tr** as well. That's why **filter0** and **filter1** would be my choices for users.

(2)

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

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

```
## [1] 0.082397
## [1] 0.02122347
```

err1. Generalization error means error on unseen data. In the two filter models mentioned above, only **filter1** is used to obtain the error on a separate test set.

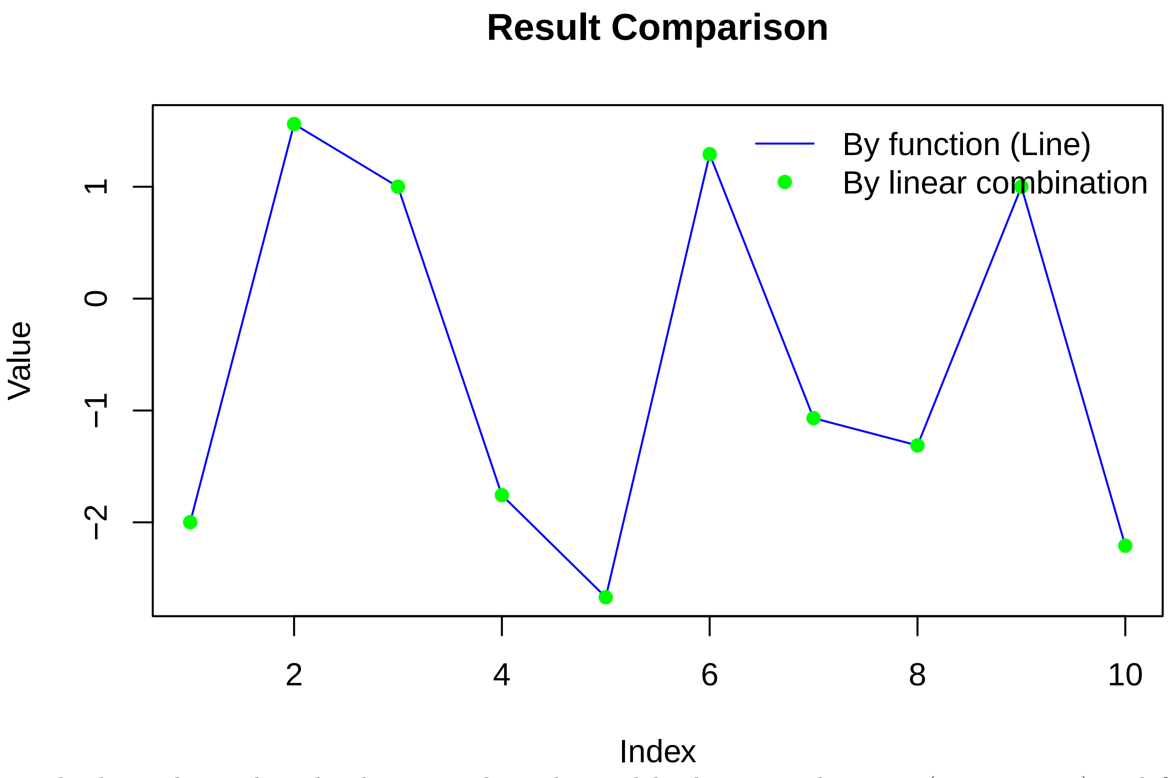
(3)

The equation for computing the decision value k_i is given as:

$$k_i = \sum_{j=1}^{n} \theta_j e^{-\sigma \|x_j - x_i\|^2} - b$$

where,
n is the number of support vectors;
θ_j is the coefficient of j^{th} support vector;
σ is the hyperparameter set to 0.05 here;
x_j is the j^{th} support vector;
x_i is the i^{th} desired point;
b is the intercept of the linear combination.

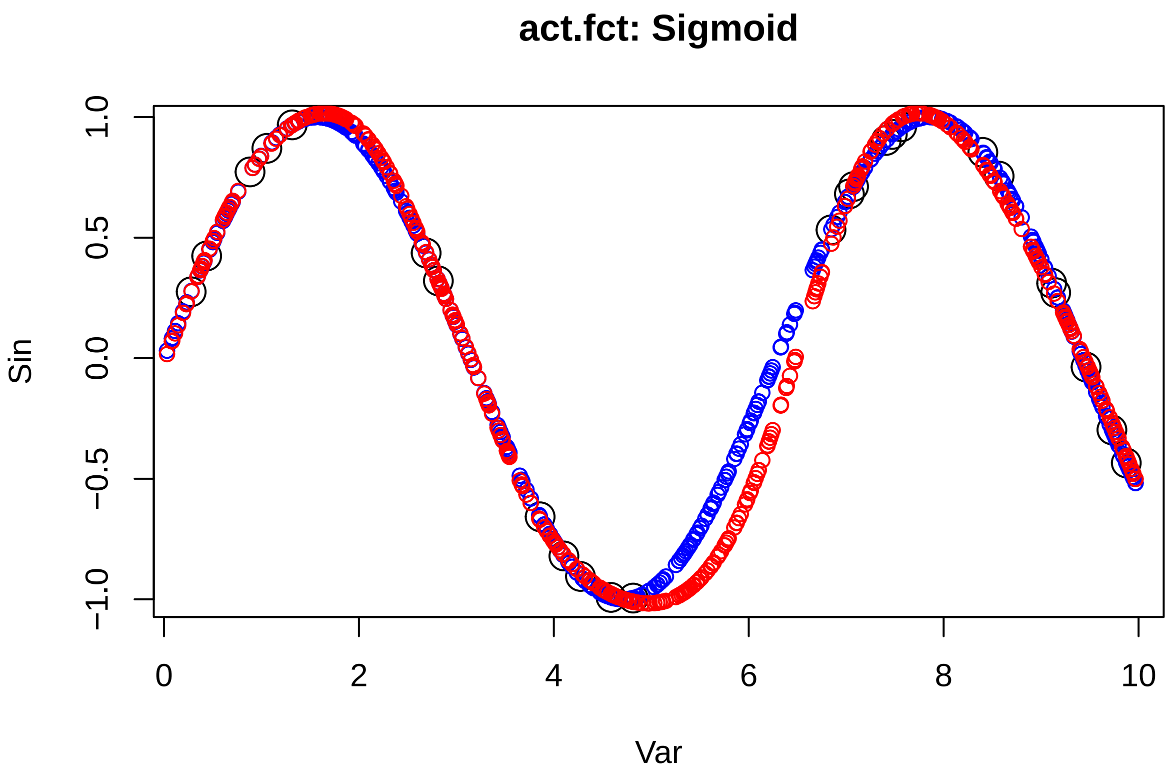
```
## Results from linear combination:

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



The graph above shows that the decision values obtained by linear combination (green points) and function **predict** (blue line) are almost the same.

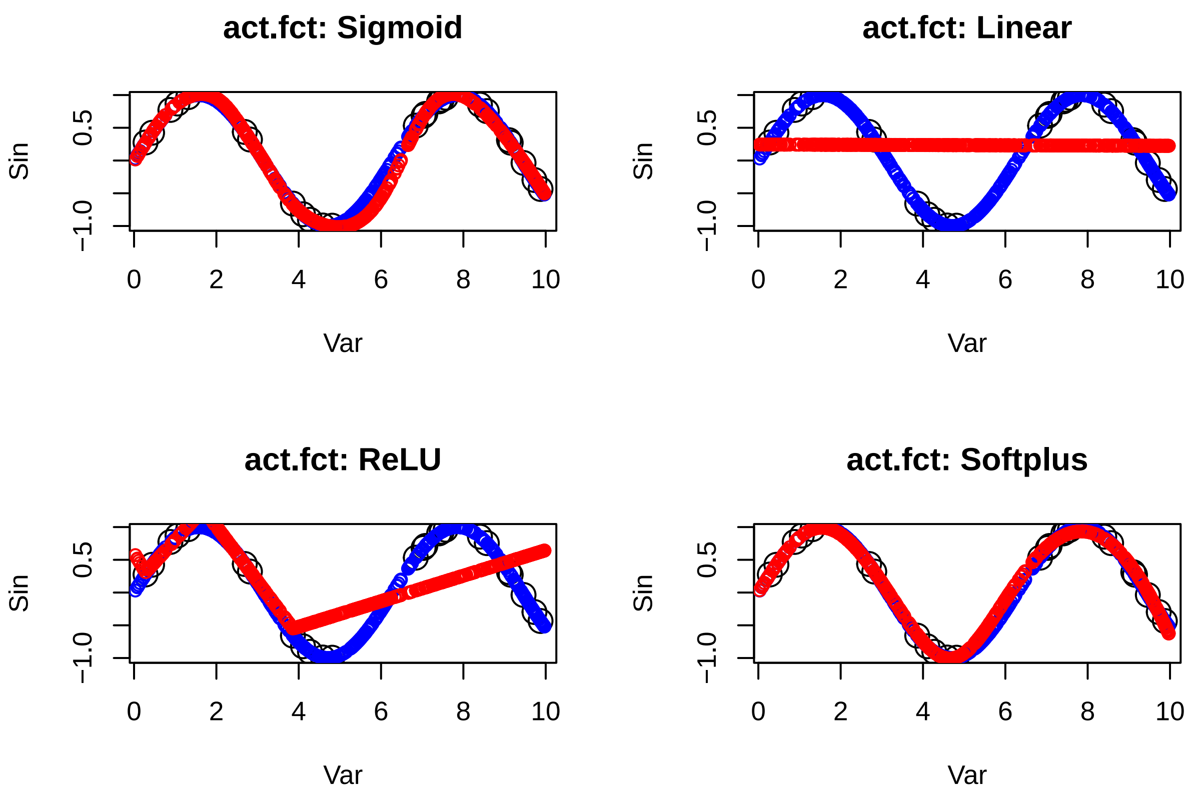
Assignment 3. NEURAL NETWORKS

(1)



According to the plot, it can be seen that prediction data are almost matched with the real values, only the deviance in the range from 5 to 7 is relatively larger than other part where fits well. Overall, the neural network model can predict sin(x) from x well with 10 hiden units.

(2)

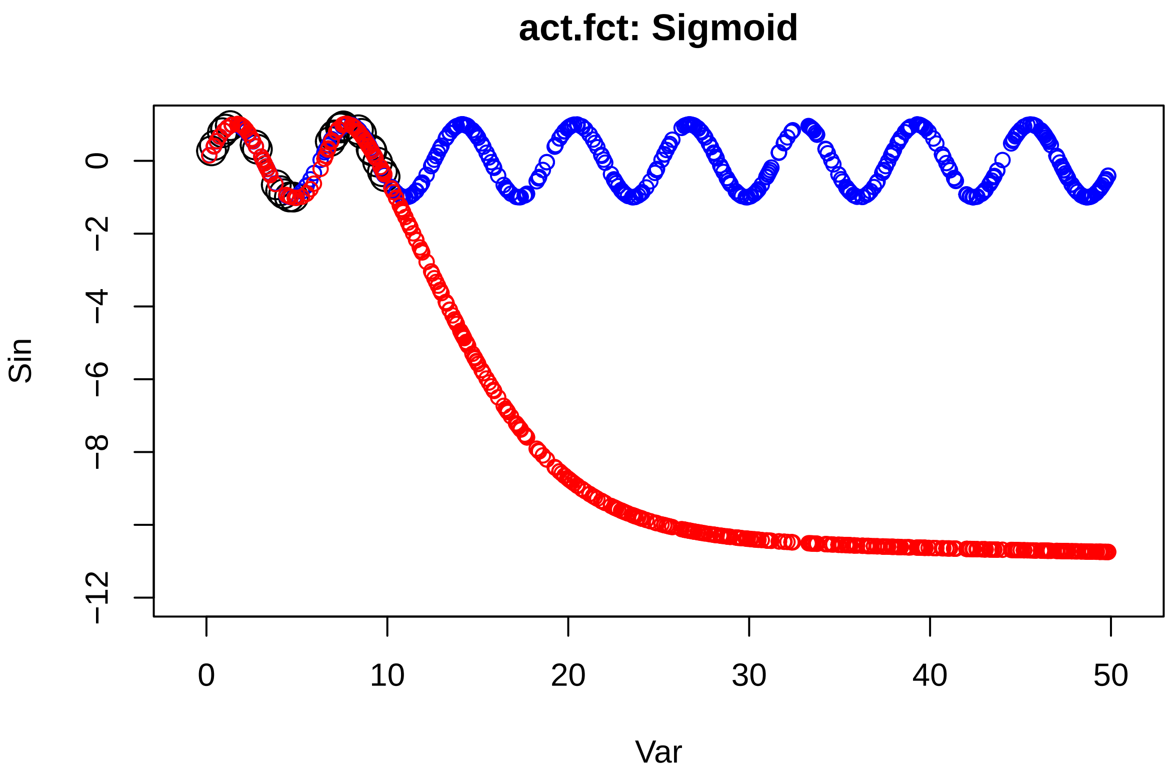


According to the plots: The flat line for h1 indicates that the neural network is not learning complex patterns. Linear activation functions are not suitable for capturing non-linear relationships, and the model essentially behaves like a linear regression.

The predicted values from h2 are closer to the real testing values, suggesting that ReLU is able to capture some non-linear features. However, the straighter line indicates that the model might not be capturing more intricate patterns in the data.

The predicted values from h3 are nearly matched with the real testing values. Softplus provides a smooth, differentiable activation function, allowing the model to learn more complex relationships in the data.

(3)



The model fits well in the interval [0,10]. Beyond 10, the prediction decreases gradually and converges to a certain value. In one word, it doesn't perform well on data outside the training range.

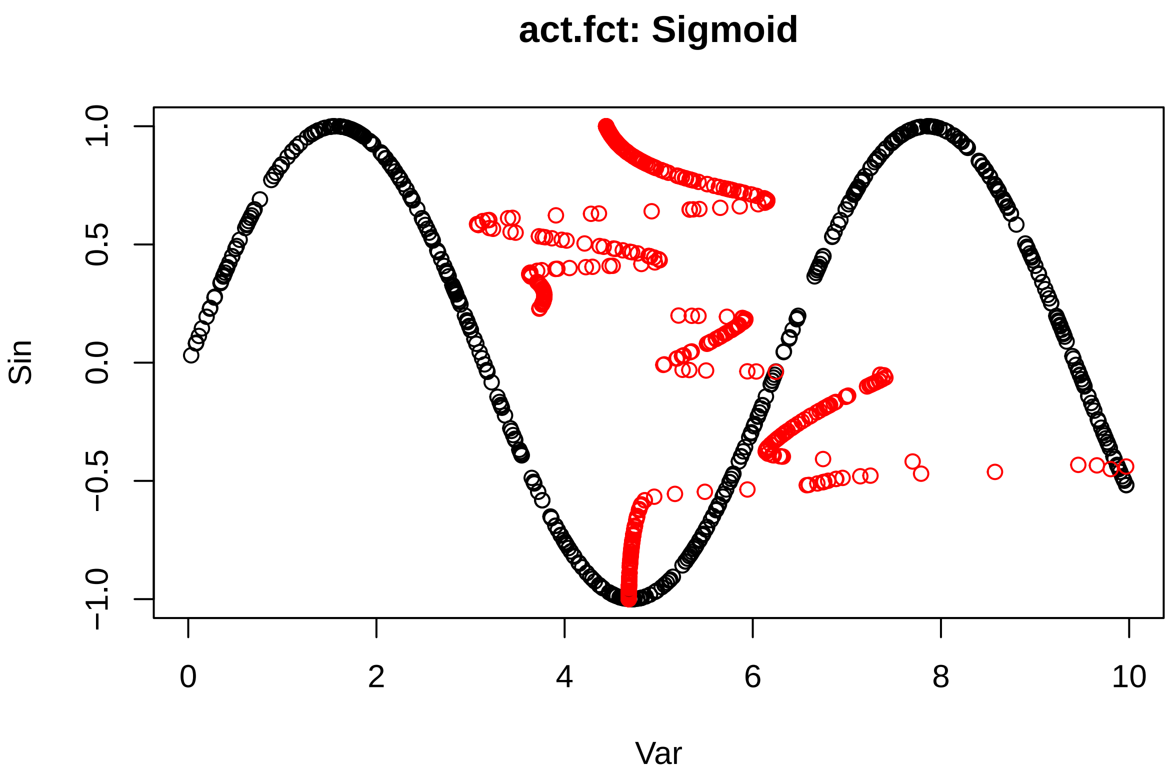
(4)

```
## Weights of nn model:

## [[1]]
## [[1]][[1]]
##            [,1]       [,2]       [,3]       [,4]      [,5]       [,6]      [,7]
## [1,] -11.870831 -0.9153178 -3.0316259  1.5313105  6.555437 -11.768525  1.397325
## [2,]   4.042494 -0.5368677  0.2603355 -0.5487656 -2.233839   1.787862 -1.259834
##            [,8]       [,9]       [,10]
## [1,]  0.2032402  0.2948804 -0.21326487
## [2,] -2.1974036 -0.5886005 -0.03177278
##
## [[1]][[2]]
##              [,1]
##  [1,]  -0.1088032
##  [2,]   0.7716196
##  [3,]  -1.0436754
##  [4,] -16.0529801
##  [5,]  -1.2718986
##  [6,]   3.2805756
##  [7,]   4.3076898
##  [8,]   0.1220703
##  [9,]  -1.2234185
## [10,]  -2.7618205
## [11,]   2.3696931
```

When train the network on data in the range [0, 10] and then test it on data in the range [0, 50], the inputs to the neurons in the hidden layer for the test data are likely to be much larger than those for the training data. This is because the weights and biases of the network are adjusted during training to produce the correct output for the training data, and these adjustments depend on the range of the training data.

When these same weights and biases are applied to the test data, the larger inputs result in the neurons in the hidden layer saturating (the first neural network model is using the default logistic (sigmoid) activation function). This results in the output of the network converging to a certain value(-11), as the contribution of the saturated neurons to the output becomes constant.

(5)



According to the plot, the predicted values didn't fit well with the real values. This is likely because when predicting x from sin(x), the relationship becomes many-to-one, as multiple x values can correspond to the same sin(x) value. This inversion introduces ambiguity, making it a more challenging problem. Also, inverting function leads to convergence issues, as the neural network struggles to find a stable mapping from the output space (sin(x)) back to the input space (x). It might get stuck in suboptimal solutions or fail to converge altogether.

Appendix

Code for Assignment 1

```r
set.seed(1234567890)
library(geosphere)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

gau_kernel <- function(dist_h, h) {
  exp(-(dist_h^2) / (2 * h^2))
}

a <- 58.4274
b <- 14.826
interest_date <- "2013-11-04"
interest_time <- c("04:00:00","06:00:00","08:00:00","10:00:00","12:00:00","14:00:00","16:00:00","18:00:0
h_distance <- 300000
h_date <- 6000
h_time <- 4

# plot the kernel value as a function of distance
par(mfrow=c(1,3))
# Plot 1: Physical Distance
dist_physical <- distHaversine(cbind(st$longitude, st$latitude), c(b, a))
plot(dist_physical, gau_kernel(dist_physical,h_distance), xlab="Physical Distance", ylab="Value", main=
# Plot 2: Day Distance
dist_date <- as.numeric(as.Date(st$date) - as.Date(interest_date))
plot(dist_date, gau_kernel(dist_date,h_date), xlab="Day Distance", ylab="Value", main="Origin: as templa
# Plot 3: Time Distance
dist_time <- as.numeric(as.POSIXct(st$time, format="%H:%M:%S")-as.POSIXct(interest_time[5], format="%H:%
plot(dist_time, gau_kernel(dist_time,h_time), xlab="Time Distance", ylab="Value", main="Origin: 12:00:00


temp_add <- vector(length=length(interest_time))
temp_mul <- vector(length=length(interest_time))

for (i in seq_along(interest_time)) {
  st$datetime <- as.POSIXct(paste(st$date, st$time), format="%Y-%m-%d %H:%M:%S")
  interest_datetime <- as.POSIXct(paste(interest_date, interest_time[i]), format="%Y-%m-%d %H:%M:%S")
  subset_data <- subset(st, st$datetime <= interest_datetime)

  dist_physical <- distHaversine(cbind(subset_data$longitude, subset_data$latitude), c(b, a))
  dist_date <- as.numeric(as.Date(subset_data$date) - as.Date(interest_date))
  dist_time <- as.numeric(as.POSIXct(subset_data$time, format="%H:%M:%S")-as.POSIXct(interest_time[i], f

  kernel_values_add <- gau_kernel(dist_physical, h_distance) + gau_kernel(dist_date, h_date) + gau_kerne
  kernel_values_mul <- gau_kernel(dist_physical, h_distance) * gau_kernel(dist_date, h_date) * gau_kerne

  temp_add[i] <- sum(subset_data$air_temperature * kernel_values_add) / sum(kernel_values_add)
  temp_mul[i] <- sum(subset_data$air_temperature * kernel_values_mul) / sum(kernel_values_mul)
}

cat("Temp (the kernel is the sum of three Gaussian kernels):\n")
temp_add
cat("Temp (the kernel is the product of three Gaussian kernels):\n")
temp_mul

par(mfrow=c(1,1))
x_label <- seq(4,24,by=2)
plot(temp_add, type="o", col="blue", pch=16, ylim=range(c(temp_add, temp_mul)), xlab="Time", ylab="Tempe
lines(temp_mul, type="o", col="green", pch=16)
axis(1, at = seq_along(x_label), labels = x_label, )
```

```r
legend("topright", legend=c("Summation", "Product"), col=c("blue", "green"), lty=1, pch=16)
```

Code for Assignment 2

```r
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))
}

filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALS
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,scaled=FALS
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,scaled=FA
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,scaled=FA
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3
```

```r
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FA

sv <- alphaindex(filter3)[[1]]
co <- coef(filter3)[[1]]
```

```r
inte <- -b(filter3)
k<-NULL
for(i in 1:10){
  k2<-NULL
  for(j in 1:length(sv)){
    dis <- spam[sv[j],-58] - spam[i,-58]
    exp_value <- exp(-0.05*sum(dis^2))
    k2 <- c(k2, exp_value)
  }
  k<-c(k, sum(co*k2)+inte)
}

cat("Results from linear combination:\n")
k
real_value <- predict(filter3,spam[1:10,-58], type = "decision")

plot(real_value, xlab = "Index", ylab = "Value", main = "Result Comparison", type = "l", col = "blue")
points(k, col = "green", pch = 16)
legend("topright", legend = c("By function (Line)", "By linear combination"), col = c("blue", "green"),
```

Code for Assignment 3

```r
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]
winit <- runif(2*10+11, -1, 1)
nn <- neuralnet(Sin ~ Var, data = tr, hidden = c(10),  startweights = winit)
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2, main="act.fct: Sigmoid")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
h1 <- function(x) x
h2 <- function(x) ifelse(x>0,x,0)
h3 <- function(x) log(1+exp(x))

nn_linear <- neuralnet(Sin ~ Var, data = tr, hidden = c(10),  startweights = winit, act.fct = h1)
nn_relu <- neuralnet(Sin ~ Var, data = tr, hidden = c(10),  startweights = winit, act.fct = h2)
nn_softplus <- neuralnet(Sin ~ Var, data = tr, hidden = c(10),  startweights = winit, act.fct = h3)

par(mfrow=c(2,2))
plot(tr, cex=2, main="act.fct: Sigmoid")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)

plot(tr, cex=2, main="act.fct: Linear")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_linear,te), col="red", cex=1)
```

```r
plot(tr, cex=2, main="act.fct: ReLU")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_relu,te), col="red", cex=1)

plot(tr, cex=2, main="act.fct: Softplus")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_softplus,te), col="red", cex=1)

par(mfrow=c(1,1))
set.seed(1234567890)

Var <- runif(500, 0, 50)
te <- data.frame(Var, Sin=sin(Var))

plot(tr,cex=2,main="act.fct: Sigmoid",xlim=c(min(Var)-1,max(Var)+1),ylim=c(-12,1))
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
cat("Weights of nn model:\n")
nn$weights
nn <- neuralnet(Var ~ Sin, data = mydata, hidden = c(10),  startweights = winit, threshold = 0.1)

plot(mydata, cex=1, main="act.fct: Sigmoid")
points(predict(nn,mydata),mydata[,2], col="red", cex=1)
```