Lab 3

Aron, Sergey, Shahin

2025-12-15

Statement of Contribution

Assignment 2 was mainly contributed by Aron. Assignment 3 was mainly contributed by Shahin. Assignment 4 was mainly contributed by Sergey. Theory questions: Q1: Aron, Q2: Shahin, Q3: Sergey.

ASSIGNMENT 1. THEORY

Q1: What is the kernel trick? Page 194: If x is used in the model only as dotproducts, we can use the kernel for phi(x)phi(x)^T. The kernel could be easier to calculate, the polynomial basis has closed form so we do not need to project into higher-dimensional space.

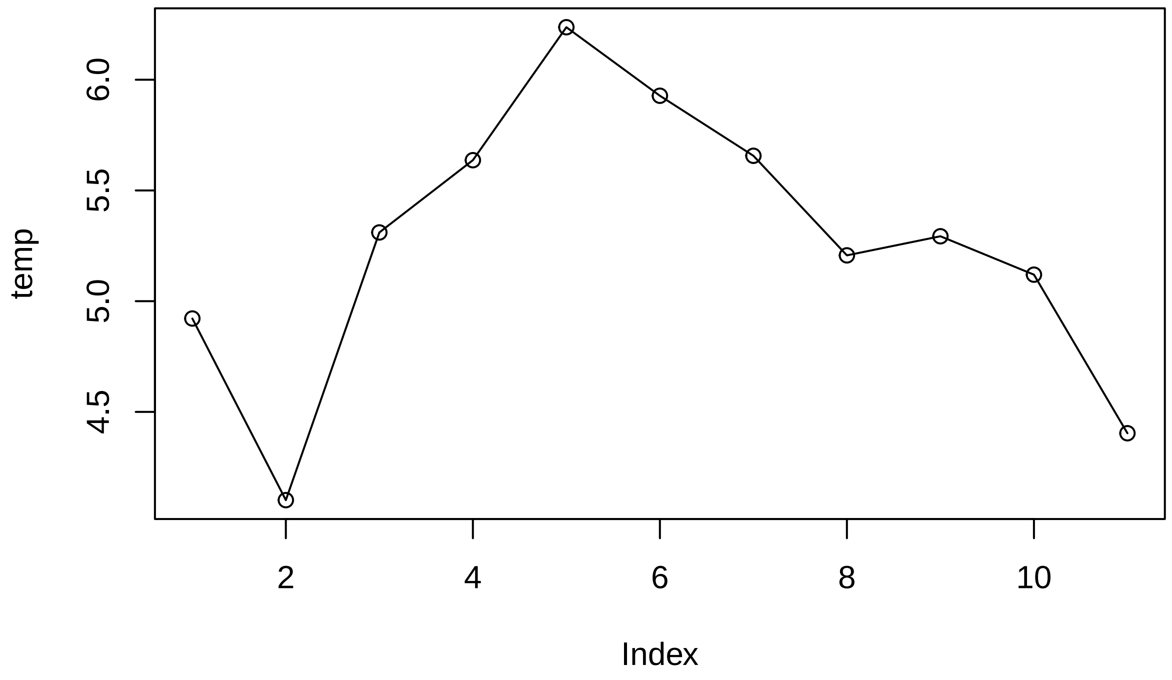
Q2: In the literature, it is common to see a formulation of SVMs that makes use of a hyperparameter C. What is the purpose of this hyperparameter? (page 211) In SVMs, the hyperparameter C controls the strength of regularisation, i.e. the trade-off between maximising the margin and penalising classification errors. A large C puts a high penalty on misclassification, forcing the model to fit the training data more closely (smaller margin, risk of overfitting). A small C allows more violations of the margin, leading to a wider margin and better generalisation.

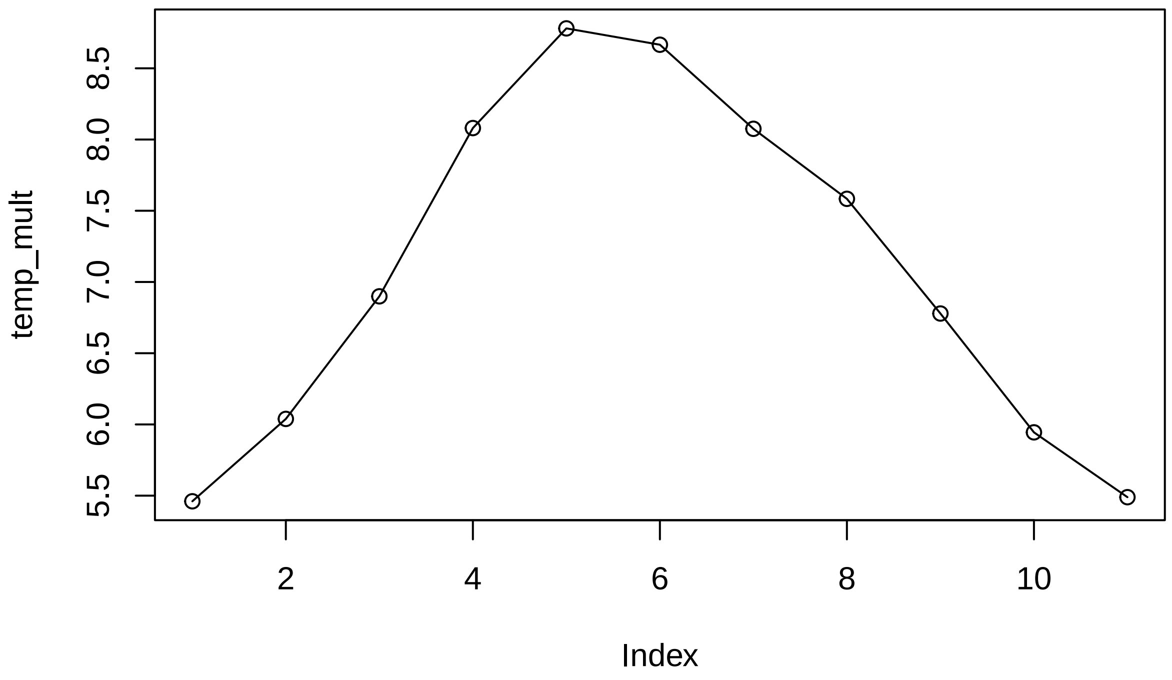
In the book, C is introduced as an alternative parameterisation of the regularisation parameter.

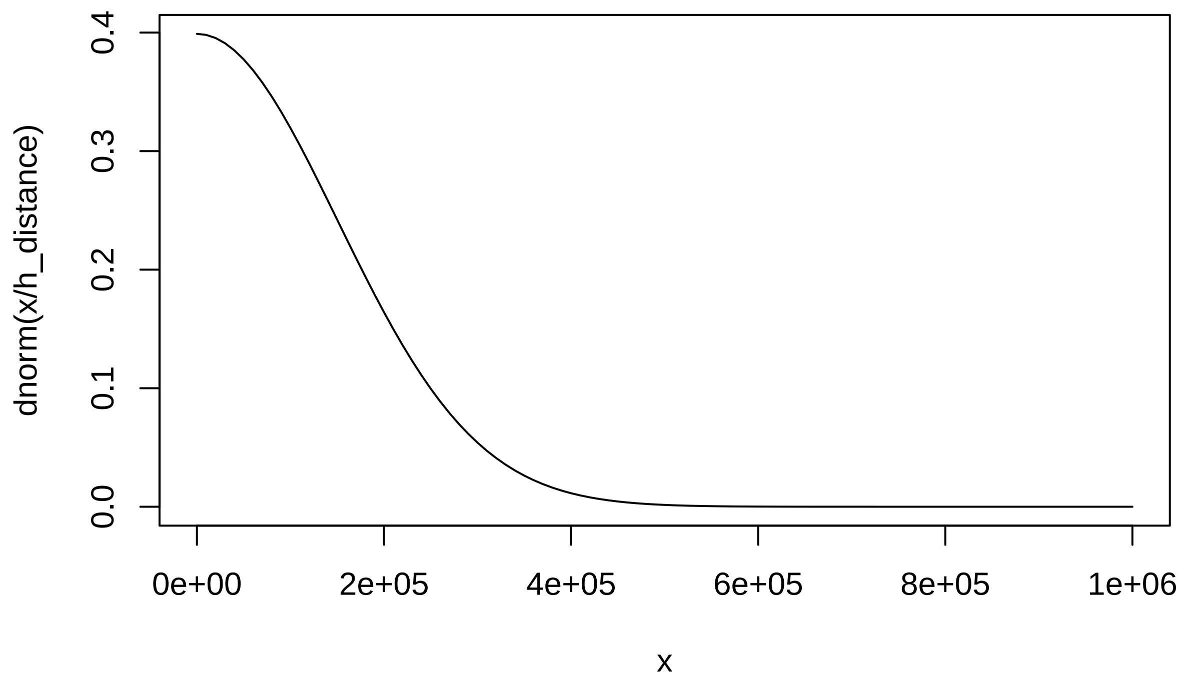
$C = \frac{1}{2*\lambda}$ or $C = \frac{1}{2*n\lambda}$

Q3: In neural networks, what do we mean by mini-batch and epoch? Page 125: A *mini-batch* is a small subset of the full dataset, typically containing $n_b = 10$, $n_b = 100$, or $n_b = 1{,}000$ observations.

A single complete pass through the entire training dataset is referred to as an *epoch*. Each epoch therefore consists of n/n_b iterations.


temp
4.5
5.0
5.5
6.0
2
4
6
8
10
Index


temp_mult
5.5
6.0
6.5
7.0
7.5
8.0
8.5
2
4
6
8
10
Index



The widths are picked like this, smaller h means we put less impact on points further away. It is scaled by standard deviation, otherwise everything gets pushed to 0 for distance for example.

h_distance <- sd(prior_df$distance) * 0.1

h_date <- sd(prior_df$day_diff) * 0.1

h_time <- sd(prior_df$delta_t) * 0.1

h had to be increased for the multiplicative kernel as it caused division by 0 issues.

The weight from the kernel decays with distance.

It is worth noting that these predictions are very poor, I tried a data in the summer and got values below 10 degrees. The choice of kernel doesn't handle seasonality but does handle trend effects.

The difference between the multiplicative and the additive kernel is that for the multiplicative kernel the deviation of one variable does not impact the overall kernel that much as for the additive kernel.

ASSIGNMENT 3. SUPPORT VECTOR MACHINES

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

The error rate rates of 4 filters are as follows.

```
## error rate of filter0 is 6.75 %
```

```
## error rate of filter1 is 8.489388 %
```

```
## error rate of filter2 is 8.2397 %
```

```
## error rate of filter3 is 2.122347 %
```

What each filter does - filter0 - Trained on tr and Evaluated on va - Used only for model selection

- filter1
  - Trained on tr and Tested on te
  - Wastes validation data for training
- filter2
  - Trained on tr + va and Tested on te
  - Does not Waste validation data for training
- filter3
  - Trained on all data including test
  - Causes data leakage
  - Gives over-optimistic results

Therefore, We return ***filter2*** because after choosing hyperparameter, C using the validation data, it retrains the model using all available non-test data.

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

Generalization error estimates how well do we think that deployed model will perform on new, unseen data.The estimate of the generalization error is ***err2***. Because

- err2 measures performance on test data
- Test data was never used during training
- Therefore, it gives an unbiased estimate of future performance

3.3 Implementation of SVM predictions.

In this assignment, the Support Vector Machine (SVM) uses the radial basis function (RBF) kernel, defined as

$$K(x, x_i) = \exp\left(-\sigma \|x - x_i\|^2\right)$$

where:
- x is the i-th point in the spam dataset (new point)
- x_i is the j-th support vector
- σ is 0.05 (given)

This kernel measures how similar the new point is to each support vector: the closer they are, the larger the kernel value.

The SVM decision function is computed as a linear combination of these kernel values:

$$f(x) = \sum_{i=1}^{n} \alpha_i \, K(x, x_i) + b$$

where:
- α_i are the linear coefficients for support vectors
- b is the intercept of the linear combination

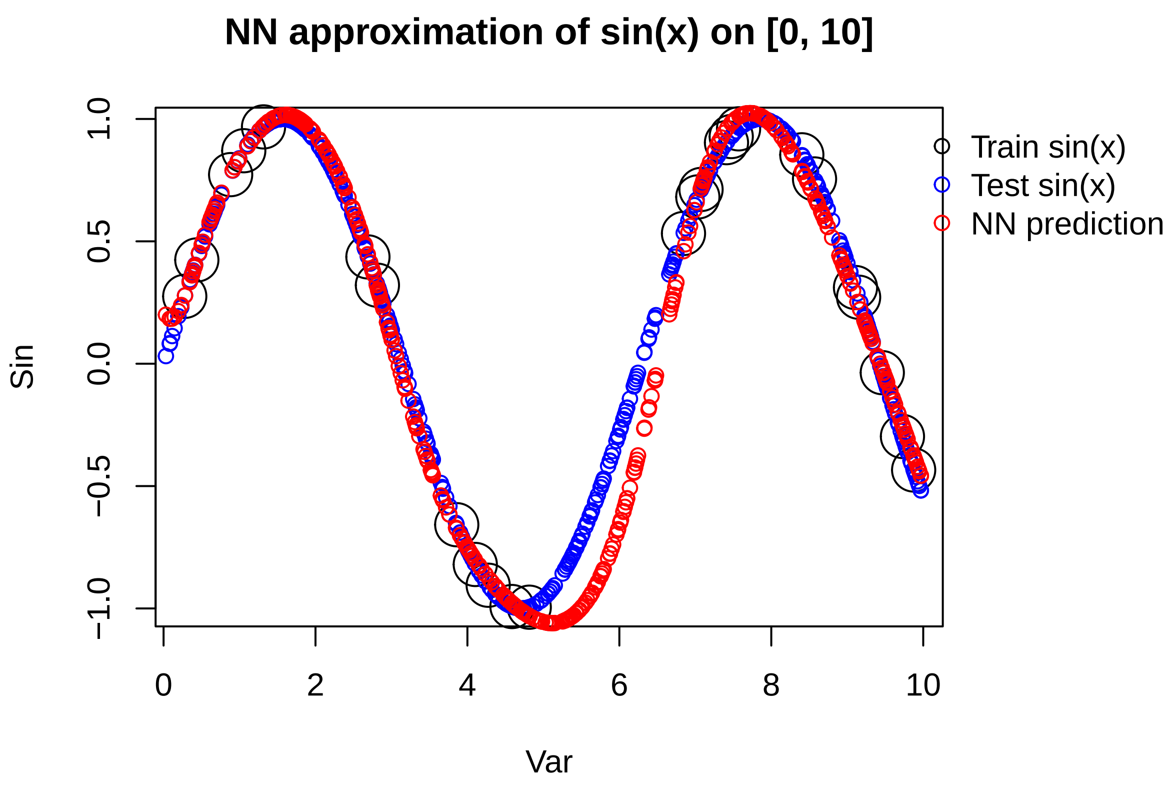
The decision function takes the similarity scores from the kernel (between the new point and each support vector), multiplies each by its weight α_i, adds them all up, and then adds the intercept b. If the total is positive, the point is classified as spam; if negative, it's non-spam.

```
##    ManualDecision SVMDecision
## 1      -1.998999   -1.998999
## 2       1.560584    1.560584
## 3       1.000278    1.000278
## 4      -1.756815   -1.756815
## 5      -2.669577   -2.669577
## 6        1.291312    1.291312
## 7       -1.068444   -1.068444
## 8       -1.312493   -1.312493
## 9        1.000184    1.000184
## 10      -2.208639   -2.208639
```

The manually computed decision values match exactly with the output of the ***predict*** function when using ***type = "decision"***, confirming the correctness of the calculations.

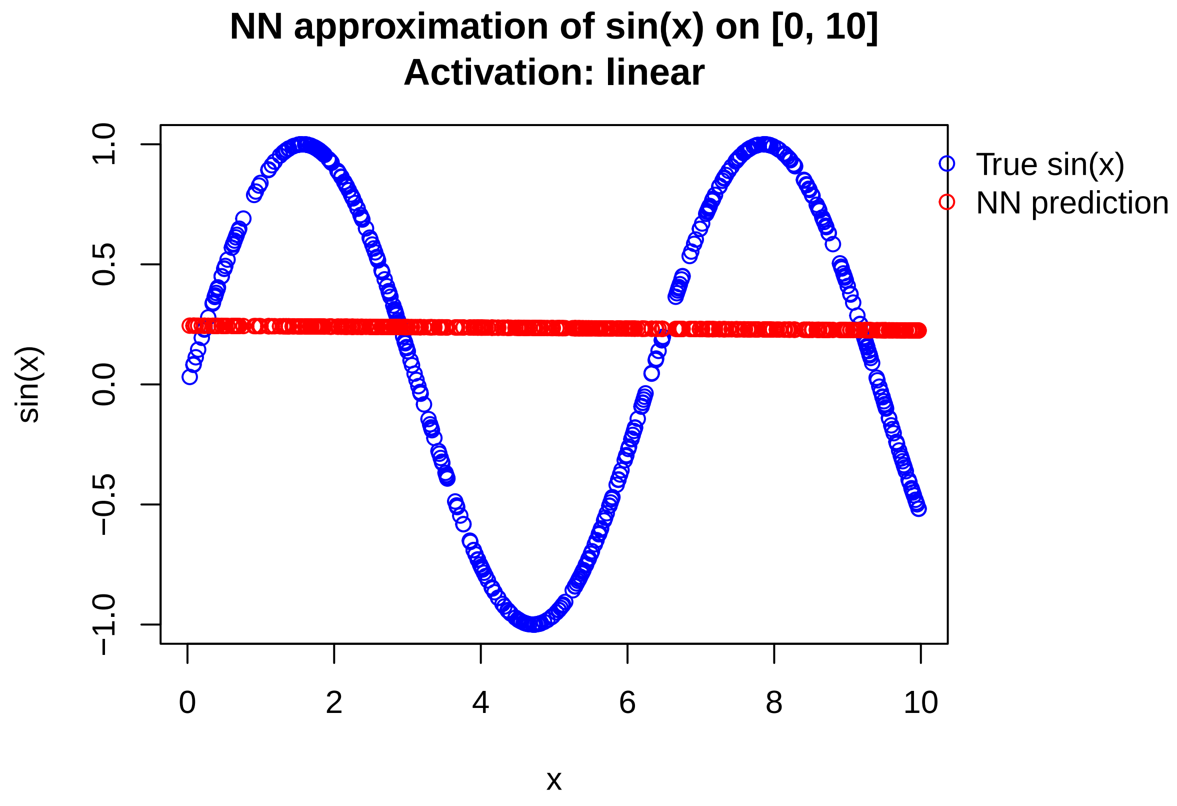
ASSIGNMENT 4. NEURAL NETWORKS

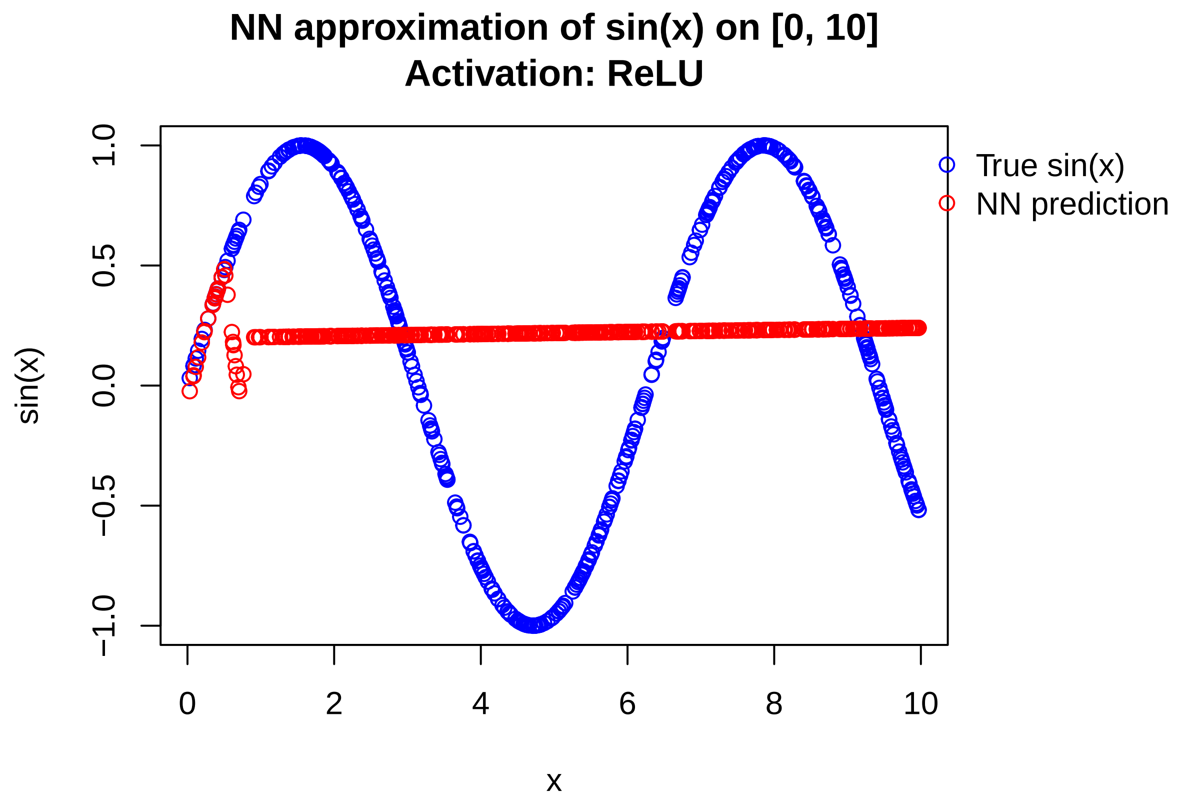
Task 1

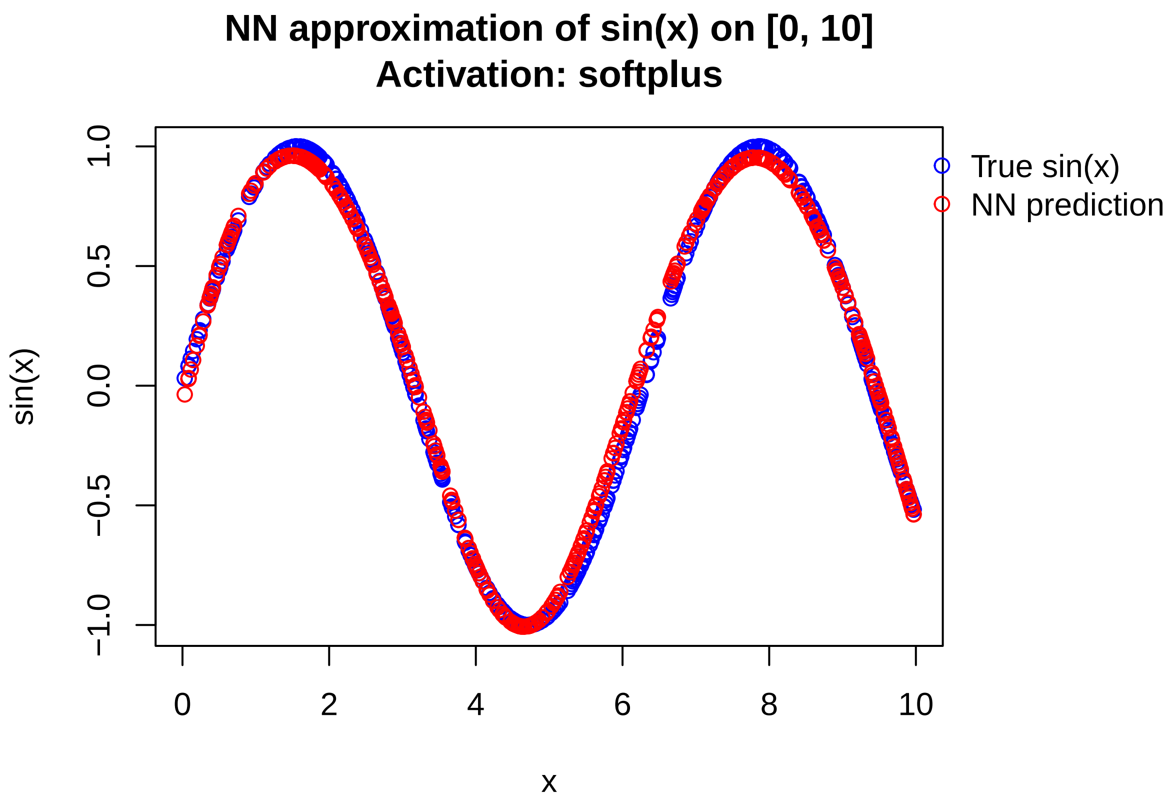


From the plot above we can see that the neutral network is able to approximate the sine function quite well in the input range of 0 to 10 even with the limited amount of training data provided (25 data points).

Task 2

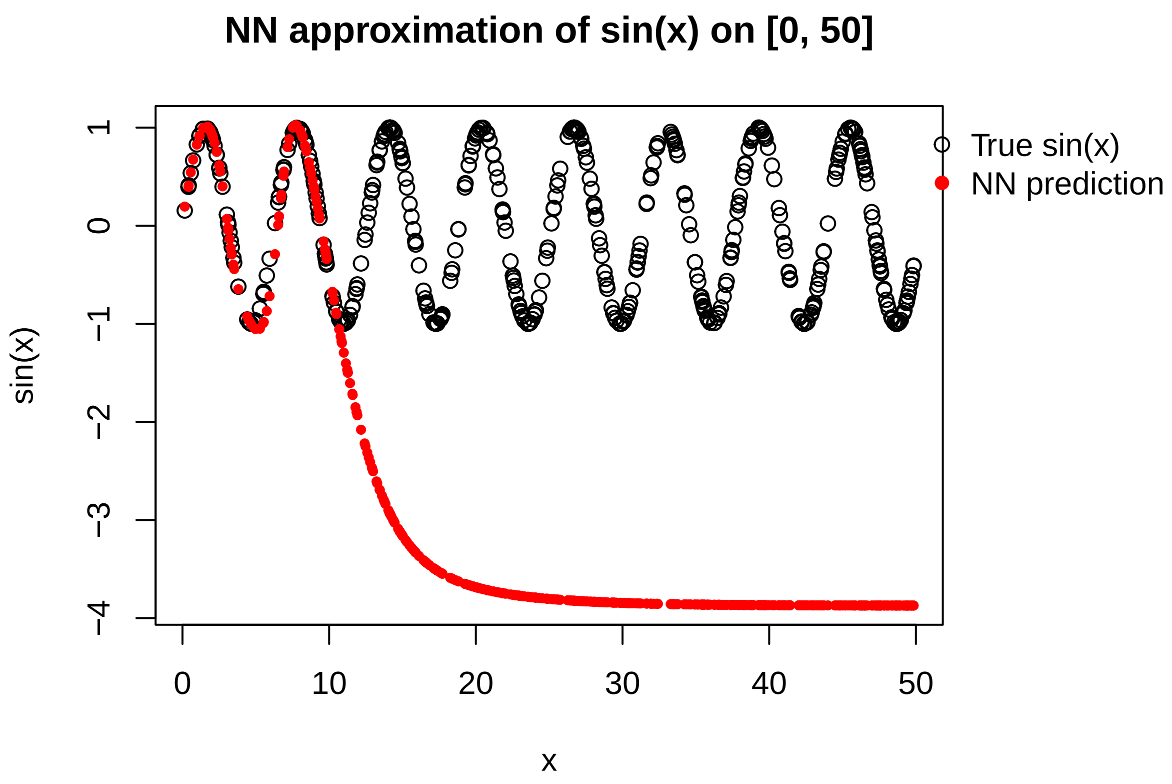

NN approximation of sin(x) on [0, 10]
Activation: linear
True sin(x)
NN prediction
sin(x)
x
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10


NN approximation of sin(x) on [0, 10]
Activation: ReLU
True sin(x)
NN prediction
sin(x)
x
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10



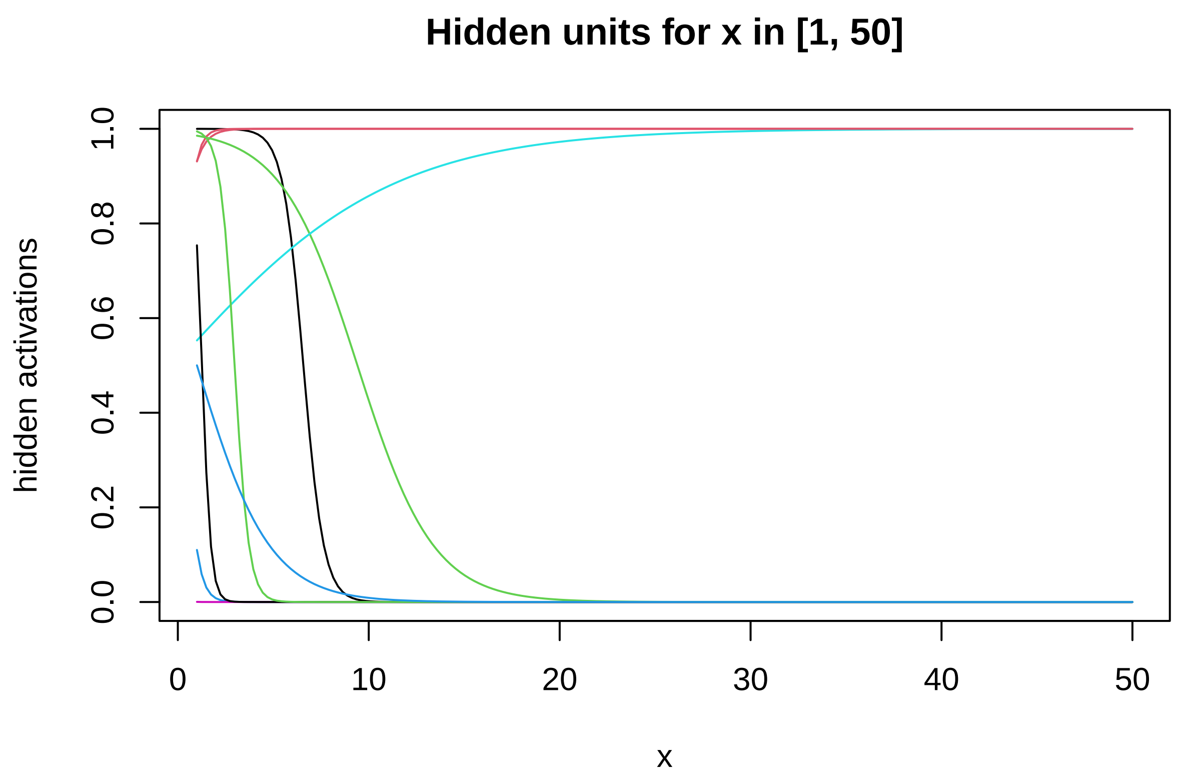
From the plots above we can see that the neural network is unable to learn the sine function at all when using the identity function and provides constant output. For the ReLU function, the function is approximated only within a very limited range and as a straight line rather than a curve. The softplus function achieves a potentially better approximation that the default logistic function to which it is closely related as the logistic function is the derivative of softplus.

Task 3



From the plot above, we can see that in the range of input values of 0 to 50, the approximation quickly breaks down when the input values exceed 10 and eventually the outputs collapse to a constant.

Task 4

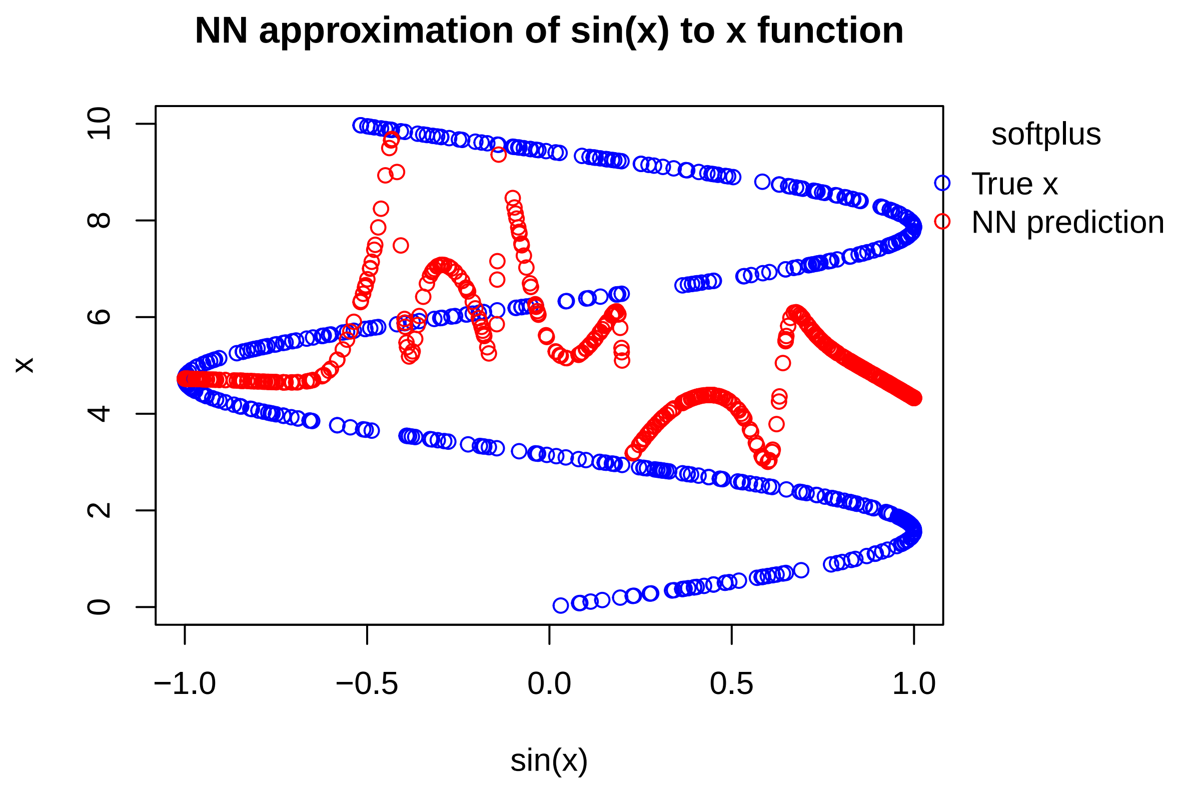


Plotting the outputs of the hidden layer demonstrates that, as the input value increases, the neurons of the hidden layer become “saturated” i.e. more and more of them are stuck at the value of either 0 or 1. When it finally happens for all neurons - on the graph above at a value of approximately 30 - the neural network will produce the same output value regardless of the input value.

It is mentioned in the literature[1] that non-periodic activation functions are not suitable for modeling the behavior of periodic functions such as sine.

[1] https://papers.neurips.cc/paper_files/paper/2020/file/1160453108d3e537255e9f7b931f4e90-Paper.pdf

Task 5



We are seeing a poor approximation because the neural network is trying to find a 1-to-1 mapping between the input and the output values but in the case of the $y = \arcsin(x)$ function it does not exist.

Appendix

```
########################
#
# ASSIGNMENT 2 CODE
#
########################
set.seed(1234567890)
library(geosphere)
stations <- read.csv("data/stations.csv", fileEncoding = "latin1")
temps <- read.csv("data/temps50k.csv")
st <- merge(stations,temps,by="station_number")
h_distance <- # These three values are up to the students
h_date <-
h_time <-
#this is linköping
latitude <- 58.4274 # The point to predict (up to the students)
longitude <- 14.826

# documentation says first one is longitude
# distance is in meters, i checked externally
distance_to_station <- sapply(1:nrow(st), function(x) distHaversine(c(st$longitude[x], st$latitude[x]),
```

```r
st$distance <- distance_to_station


st$date <- st$date |> as.Date()


# christmas 2008
date_of_interest <- "2008-12-24" |> as.Date()

times <- seq(4, 24, by = 2) |> as.character()
times <- sapply(1:length(times),
                function(x) {
  if(nchar(times[x]) == 1){return(paste0("0", times[x]))}
                return(times[x])}
  )
times <- paste0(times, ":00:00")
temp <- vector(length=length(times))
temp_mult <- vector(length=length(times))

for (i in seq_along(times)) {
  hour_of_interest <- times[i]
  numeric_hour_of_interest <- substr(hour_of_interest, 1, 2) |> as.numeric()

date_diff <- difftime(date_of_interest, st$date) |> as.numeric()
st$day_diff <- date_diff

# negative happen after our date_of_interest
posterior_index1 <- date_diff < 0


# inefficient code but sufficient, if clean use lubridate
same_day <- st$date == date_of_interest
hour_later <- (substr(st$time, 1, 2) |> as.numeric()) > numeric_hour_of_interest

posterior_index2 <- hour_later & same_day

posterior_index <- posterior_index1 | posterior_index2


prior_df <- st[!posterior_index, ]


# time is cyclic, so if we want to have hour of day as a variable,
# similar times to 23 are both 21 and 01...
hour_fun <- function(x, y) {
  delta <- abs(x - y)

  if (delta <= 12) {
    return(delta)
  } else {
    if (x < y) {

      return( abs((x + 24) - y) )
    } else {
      return( abs((y + 24) - x) )
    }

  }

}


prior_df$delta_t <- sapply(1:nrow(prior_df), function(x) hour_fun(numeric_hour_of_interest, prior_df$ti

# small h makes the distance larger
h_distance <- sd(prior_df$distance) * 0.1
h_date <- sd(prior_df$day_diff) * 0.1
h_time <- sd(prior_df$delta_t) * 0.1

# too large h means it converges to the mean
# mean(prior_df$air_temperature)

a <- sum(dnorm(prior_df$distance / h_distance) * prior_df$air_temperature) + sum(dnorm(prior_df$day_dif

b <-  sum(dnorm(prior_df$distance / h_distance)) + sum(dnorm(prior_df$day_diff / h_date)) + sum(dnorm(pr

prediction <- a/b


h_distance <- sd(prior_df$distance) * 0.5
h_date <- sd(prior_df$day_diff) * 0.5
h_time <- sd(prior_df$delta_t) * 0.5

c <- dnorm(prior_df$distance / h_distance) * dnorm(prior_df$day_diff / h_date) * dnorm(prior_df$delta_t


prediction_mult <- sum(c * prior_df$air_temperature) / sum(c)

temp[i] <- prediction
temp_mult[i] <- prediction_mult

# loop end
# -------------------
}

plot(temp, type="o")
plot(temp_mult, type="o")

curve(dnorm(x/h_distance), from = 0, to = 1000000)

#######################
#
# ASSIGNMENT 3 CODE
#
#######################
library(kernlab)
```

```r
set.seed(1234567890)
############################  Load Data and split data sets #####################
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, ]

####################  Get validation error and corresponding C #####################
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))
}

c1 <- which.min(err_va)

############################ filter0,filter1,filter2,filter3 #####################
filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",
               kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)

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

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


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

cat("error rate of filter0 is", (err0)*100,"%\n")
cat("error rate of filter1 is", (err1)*100,"%\n")
cat("error rate of filter2 is", (err2)*100,"%\n")
cat("error rate of filter3 is", (err3)*100,"%\n")
```

```r
###########################  Manual Computations ####################
sv <- alphaindex(filter3)[[1]]   # support vector indexes
co <- coef(filter3)[[1]]         # alpha coefficients
inte <- -b(filter3)              # intercept
sigma <- 0.05                    # RBF kernel width

manual_decision <- numeric(10)

for(i in 1:10){
  sum_k <- 0
  for(j in 1:length(sv)){
    diff <- spam[sv[j], -58] - spam[i, -58]
    k_val <- exp(-sigma * sum(diff^2))
    sum_k <- sum_k + co[j] * k_val
  }
  manual_decision[i] <- sum_k + inte
}

# --- SVM predict() results ---
svm_decision <- predict(filter3, spam[1:10, -58], type = "decision")

# --- Simple comparison table ---
decision_table <- data.frame(
  ManualDecision = round(manual_decision, 6),
  SVMDecision = round(svm_decision, 6)
)

decision_table
#######################
#
# ASSIGNMENT 4 CODE
#
#######################
# Set all variables to null to avoid possible variable name clashes between unrelated parts of the lab
rm(list = ls())

library(neuralnet)
library(sigmoid)

get_data <- function(max_value, train_set_count, total_count=500) {
  set.seed(1234567890)
  Var   <- runif(total_count, 0, max_value)
  mydata <- data.frame(Var, Sin = sin(Var))
  tr <- mydata[1:train_set_count, ]
  if (total_count > train_set_count) {
    te <- mydata[(train_set_count+1):total_count, ]
  }


  list(tr = tr, te = te)
}

train_network <- function(tr,
                          act_fct   = "logistic",
                          form      = Sin ~ Var,
                          threshold = NULL) {
  set.seed(1234567890)
  winit <- runif(21, min = -1, max = 1)

  args <- list(
    formula       = form,
    data          = tr,
    hidden        = 10,
    startweights  = winit,
    act.fct       = act_fct,
    linear.output = TRUE
  )

  if (!is.null(threshold)) {
    args$threshold <- threshold
  }

  nn <- do.call(neuralnet, args)
  nn
}
mydata <- get_data(max_value=10,train_set_count=25)
tr <- mydata$tr
te <- mydata$te
nn <- train_network(tr)

op <- par(mar = c(5, 4, 4, 8), xpd = TRUE)

plot(tr, cex = 3, main = "NN approximation of sin(x) on [0, 10]")
points(te, col = "blue", cex = 1)
points(te[, 1], predict(nn, te), col = "red", cex = 1)

legend("topright",
       inset = c(-0.3, 0),
       legend = c("Train sin(x)", "Test sin(x)", "NN prediction"),
       col    = c("black","blue", "red"),
       pch    = c(1, 1),
       bty    = "n")
h1 <- function(x) x                         # identity
# h2 <- function(x) pmax(0, x)              # ReLU
# as explained here:
# https://stackoverflow.com/questions/34532878/
# package-neuralnet-in-r-rectified-linear-unit-relu-activation-function
# it won't work as the pmax function is not differentiable, the suggestion is to use
# the relu function from the sigmoid package
h2 <- relu
h3 <- function(x) log(1 + exp(x))         # softplus


acts <- list(
  linear   = h1,
  ReLU     = h2,
  softplus = h3
)


for (name in names(acts)) {
  h <- acts[[name]]

  nn_temp    <- train_network(tr, act_fct = h)
  predictions <- as.vector(predict(nn_temp, te["Var"]))
  y_all       <- c(te$Sin, predictions)

  op <- par(mar = c(5, 4, 4, 8), xpd = TRUE)

  plot(te$Var, te$Sin,
       col = "blue", cex = 1,
       xlab = "x", ylab = "sin(x)",
       main = paste("NN approximation of sin(x) on [0, 10]\nActivation:", name),
       ylim = range(y_all))

  points(te$Var, predictions, col = "red", cex = 1)

  legend("topright",
         inset = c(-0.3, 0),
         legend = c("True sin(x)", "NN prediction"),
         col    = c("blue", "red"),
         pch    = c(1, 1),
         bty    = "n")
}
set.seed(1234567890)
Var_new  <- runif(500, 0, 50)
newdata  <- data.frame(Var = Var_new, Sin = sin(Var_new))

pred_new <- predict(nn, newdata)

y_all <- c(newdata$Sin, pred_new)

op <- par(mar = c(5, 4, 4, 8), xpd = TRUE)

plot(newdata$Var, newdata$Sin,
     xlab = "x", ylab = "sin(x)",
     main = "NN approximation of sin(x) on [0, 50]",
     ylim = range(y_all))
points(newdata$Var, pred_new,
       col = "red", pch = 16, cex = 0.7)

legend("topright",
       inset = c(-0.3, 0),
       legend = c("True sin(x)", "NN prediction"),
       col    = c("black", "red"),
       pch    = c(1, 16),
       bty    = "n")
x_big  <- seq(1, 50, length.out = 200)
dat_big <- data.frame(Var = x_big)
```

```r
# collapsing to a value around -2.785
W1 <- nn$weights[[1]][[1]]
W2 <- nn$weights[[1]][[2]]
sigm <- function(z) plogis(z)  # logistic activation

# Hidden pre-activations and activations for x
z_hidden <- outer(x_big, W1[2, ], `*`) + matrix(W1[1, ], nrow = length(x_big), ncol = 10, byrow = TRUE)
h_hidden <- sigm(z_hidden)

matplot(x_big, h_hidden, type = "l", lty = 1,
        xlab = "x", ylab = "hidden activations",
        main = "Hidden units for x in [1, 50]")
set.seed(1234567890)
Var   <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin = sin(Var))
nn_temp    <- train_network(mydata, form = Var ~ Sin, threshold = 0.1)
predictions <- as.vector(predict(nn_temp, mydata["Sin"]))
y_all       <- c(mydata$Var, predictions)

op <- par(mar = c(5, 4, 4, 8), xpd = TRUE)

plot(mydata$Sin, mydata$Var,
     col = "blue", cex = 1,
     xlab = "sin(x)", ylab = "x",
     main = "NN approximation of sin(x) to x function",
     ylim = range(y_all))

points(mydata$Sin, predictions, col = "red", cex = 1)

legend("topright",
       inset = c(-0.3, 0),
       legend = c("True x", "NN prediction"),
       col    = c("blue", "red"),
       pch    = c(1, 1),
       bty    = "n",
       title  = name)
```