report_lab3

Yanjie Lyu, Yi Yang, Qingxuan Cui

2024-12-11

Contents

1 Theory **1**

1.1 In the literature, it is common to see a formulation of SVMs that makes use of a hyper-parameter C. What is the purpose of this hyperparameter? 1

1.2 In neural networks, what do we mean by mini-batch and epoch? 2

2 Assignment 3 Support Vector Machine **2**

2.1 Which filter do we return to the user ? Why? 2

2.2 What is the estimate of the generalization error of the filter returned to the user? Why? . . . 2

2.3 Implementation of SVM predictions. 2

3 Assignment 4 **3**

3.1 Task 1 3

3.2 Task 2 3

3.3 Task 3 and 4: comment and explain the convergence 4

3.4 Task 5: Explain the prediction of x by sinx 5

4 Appendix **5**

4.1 Code for assignment 3 5

4.2 Code for Assignment 4 7

1 Theory

1.1 In the literature, it is common to see a formulation of SVMs that makes use of a hyper-parameter C. What is the purpose of this hyperparameter?

In SVM, the hyperparameter C can be expressed as $C = \frac{1}{2\lambda}$, where λ is the L_2-regularization parameter in the primal problem, used to determine the upper bound of the Lagrange multipliers, thereby constraining the model's complexity. As C increases, λ decreases, allowing for a larger θ. This results in a narrower decision boundary and subsequently reduces the number of support vectors.

(P209-P210, P215)

1.2 In neural networks, what do we mean by mini-batch and epoch?

mini-batch: mini-batch is the subset of training data randomly sampled, aiming to solve the problem of excessive computation time and memory space consumption caused by using the entire training set to train the model and updating the parameters, especially when many of them are probably relatively similar data points in the training set. A mini-batch can typically contain 1, 10 or 100 data points, but it is recommended to set it to the nth power of 2, so that it can be aligned with the memory size of the GPU.

epoch: One complete pass through the training data is called an epoch. An epoch consists of $\frac{n}{n_b}$ where n represents the size of train data, and n_b represents the batch size.

(P124-P125)

2 Assignment 3 Support Vector Machine

filter	filter0	filter1	filter2	filter3
error_rate	0.165	0.167	0.150	0.014

2.1 Which filter do we return to the user ? Why?

Filter3 should be returned to the user.

Reasons:

Filter3 is trained using the entire dataset (`spam`), which includes the training set, validation set, and test set. By leveraging all available data,Filter3 is able to utilize the maximum amount of information to build the most robust model possible.

Once the testing phase is complete and the model has been evaluated, the focus shifts to creating the best model for practical use. Using all available data to train this model is the optimal choice.

2.2 What is the estimate of the generalization error of the filter returned to the user? Why?

The error rate of filter2 should be returned to the user.

Reasons:

This ensures that the evaluation is unbiased and reflects the model's ability to generalize to unseen data.

Filter2's error is a more accurate estimate of the generalization error because the test set remains independent and has not been contaminated by the training process.

2.3 Implementation of SVM predictions.

Based on the table below, we can see the values and labels from manual prediction and function based prediction. Except the first data point, the labels are same.

	data point1	data point2	data point3	data point4	data point5	data point6	data point7	data point8	data point9	data point10
manual prediction label	1	1	1	-1	-1	1	-1	-1	1	-1
prediction label	-1	1	1	-1	-1	1	-1	-1	1	-1

3 Assignment 4

3.1 Task 1

The model predictions are very good, except for a small deviation from var=5 to var=7, which is comparable to the sin values of the test set.

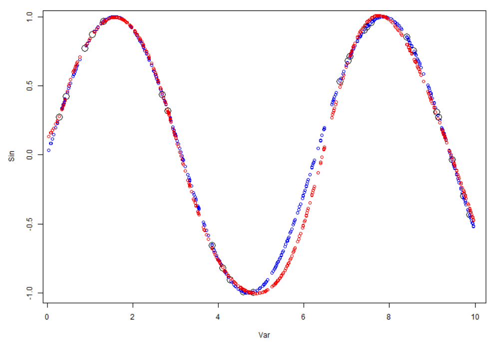

Figure 1: A.4.1: sigmoid

3.2 Task 2

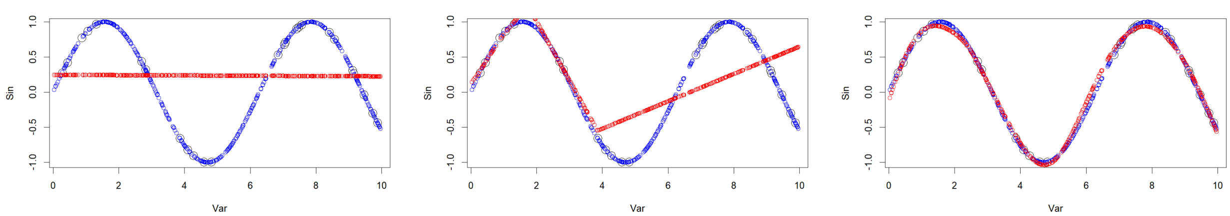

Figure 2: A.4.2: NN with different activation function

Linear function:

1. The Linear activation function is unable to introduce nonlinear transformations, resulting in the entire neural network can only behave as a simple linear mapping, equivalent to a single-layer model.
2. Due to the lack of nonlinear expression ability, the model can not capture the periodic characteristics of the sine function, and finally fits to an approximate horizontal trend line.

ReLU function:

The ReLU activation function is defined as: $h_2(x) = \max(0, x)$, In the early stage of network training, because the weight is not stable, the input value to ReLU will fluctuate back and forth in the positive and negative interval, so that some ReLU units will be activated in the positive interval (output linear segment), and some will be cut off in the negative interval (output 0). This state causes the network to generate nonlinear feature transformation and improves the fitting ability of the model.

However, at the later stage of training, if the parameters of some layers continue to learn and adjust, the input through this layer will fall into the positive semi-axis of the ReLU most of the time (that is, the x$>$0 region). In this case, the output of these ReLU units approaches the linear mapping state of y=x, and the layer's transformation of the input features no longer has the nonlinear characteristics of the previous layer. When most of the active elements of the network enter this single linear interval, the effective nonlinearity of the whole network will decrease significantly, and even approximate to a roughly linear mapping. This will cause the model to show similar linear model behavior at the later stage of training, and it is difficult to further improve the fitting ability of complex tasks.

Softplus function:

The softplus function basically matches the curve of the test set labeling.

Unlike ReLU, Softplus is smooth and differentiable everywhere, including at x=0. This property makes it more mathematically "well-behaved" in optimization tasks.

Softplus provides a small but non-zero gradient for x$<$0, allowing weights to update. Softplus offers controlled gradient flow, making it more stable in networks that require smooth transitions.

3.3 Task 3 and 4: comment and explain the convergence

To train neural networks, the data used comes from 0 to 10. Therefore, the network can only learn and fit sinusoidal function properties within this range.When the Sigmoid activation function is used, the output of the network is limited to the range (0,1). If the input x equals to 20, the value in hidden layer after the sigmoid function is:
1 1 1 1.018783e-12 1.241817e-10 9.302363e-08 0.002334578 5.432176e-14 1 8.941231e-09
It can be seen that when the input is greater than 10, the output form of the hidden layer after conversion is: the first, second, third and ninth data are infinitely close to 1, while the rest are infinitely close to 0, which leads to the subsequent output being a fixed linear transformation.
Looking at the weights of the output layer, we can see that the values of the first, second, third, and ninth weights are -1.7121097, -0.9561984, 0.1362801, -1.3999957, and the value of bias is 0.8266328. It is calculated that it will eventually converge at -3.105391

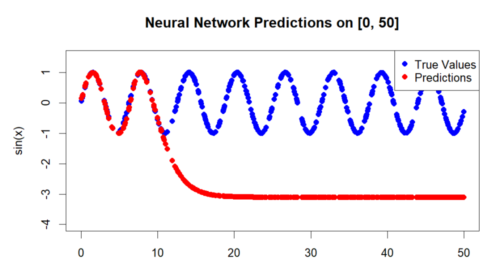

Figure 3: A.4.3: New NN plot with 500 points in the interval [0;50]

3.4 Task 5: Explain the prediction of x by sinx

The sine function is periodic, meaning the same $sin(x)$ value can correspond to multiple x values.
The neural network has only one output neuron, which makes it inherently unable to produce multiple outputs for the same input. Instead, it settles on approximating one of the valid x values for a given $sin(x)$. From the graph, we can see that the model chooses to predict the smallest possible x that corresponds to each $sin(x)$ value. This is likely because the training process minimizes the MSE loss, which encourages predictions to converge toward one specific solution rather than splitting across multiple possibilities.

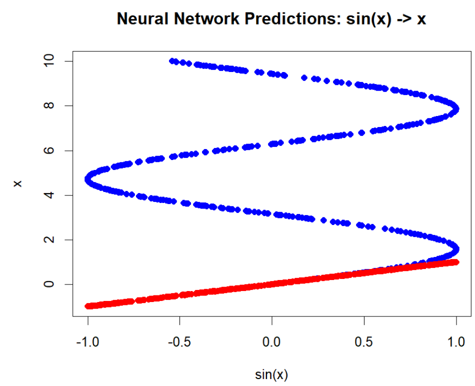

Figure 4: A.4.4: Neural Network Predictions: sin(x) -> x

4 Appendix

4.1 Code for assignment 3

```
# Lab 3 block 1 of 732A99/TDDE01/732A68 Machine Learning
# Author: jose.m.pena@liu.se
# Made for teaching purposes

library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
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))
}
# filter
# trained by train dataset
# find the optimal C on validation dataset
```

```
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)
err0
# filter0
# trained by train data
# evaluated by validation data

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)
err1
# filter1
# trained by train data
# evaluated by test data

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
# filter2
# trained by train valid data? what's that for?
# evaluated by test data

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
# filter3
# trained by whole data
# evaluated by test data

pred_vec = c()
sigma = 0.05
support_vector_indices = alphaindex(filter3)[[1]]
support_vector = spam[support_vector_indices, -58]
sv_labels = spam[support_vector_indices, 58]
sv_labels = ifelse(sv_labels == "spam", -1, 1)

coef = coef(filter3)[[1]]
intercept = - b(filter3)
x = spam[, -58]
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  k2 = 0
  for(j in 1:length(support_vector_indices)){
    rbf = exp(-sum((x[i,] - support_vector[j,])^2)/(2 * sigma^2))
    k2 = k2 + rbf * coef[j]
  }
```

```
  pred_vec=c(pred_vec, k2 + intercept)
}
pred_vec_label = ifelse(pred_vec < 0, -1, 1)


pred_func = as.vector(predict(filter3,spam[1:10,-58], type = "decision"))
pred_func_label = ifelse(pred_func < 0, -1, 1)
```

4.2 Code for Assignment 4

```
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]
n_input <- 1
n_hidden <- 10
n_output <- 1

weights_input_to_hidden <- n_input * n_hidden
weights_hidden_to_output <- n_hidden * n_output
n_weights <- weights_input_to_hidden + weights_hidden_to_output
bias_hidden <- n_hidden
bias_output <- n_output
n_weights <- n_weights + bias_hidden + bias_output
winit <- runif(n_weights, min = -1, max = 1)

nn <- neuralnet(Sin ~ Var, data = tr, hidden = 10, act.fct = "logistic", linear.output = TRUE,
                startweights = list(
                  matrix(winit[1:weights_input_to_hidden], nrow = n_input, ncol = n_hidden),
                  matrix(winit[(weights_input_to_hidden + 1):(weights_input_to_hidden + weights_hidden_
                         nrow = n_hidden, ncol = n_output)
                ))
    # Plot of the training data (black), test data (blue), and predictions (red)
    plot(tr, cex=2)
    points(te, col = "blue", cex=1)
    points(te[,1],predict(nn,te), col="red", cex=1)

    # Comment your results

# task 2

activation_functions <- list(
      linear = function(x) x,
      relu = function(x) ifelse(x > 0, x, 0),
      softplus = function(x) log(1 + exp(x))
    )
## linear
nn_lin <- neuralnet(Sin ~ Var, data = tr, hidden = 10, act.fct = activation_functions[["linear"]],
```

```
               linear.output = TRUE,startweights = list(
                 matrix(winit[1:weights_input_to_hidden], nrow = n_input, ncol = n_hidden),
                 matrix(winit[(weights_input_to_hidden + 1):(weights_input_to_hidden + weights_hidden_t
                        nrow = n_hidden, ncol = n_output)
               ))
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_lin,te), col="red", cex=1)

## relu

nn_relu <- neuralnet(Sin ~ Var, data = tr, hidden = 10, act.fct = activation_functions[["relu"]],
                    linear.output = TRUE,startweights = list(
                      matrix(winit[1:weights_input_to_hidden], nrow = n_input, ncol = n_hidden),
                      matrix(winit[(weights_input_to_hidden + 1):(weights_input_to_hidden + weights_hid
                             nrow = n_hidden, ncol = n_output)
                    ))
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_relu,te), col="red", cex=1)

## softplus

nn_softplus <- neuralnet(Sin ~ Var, data = tr, hidden = 10, act.fct = activation_functions[["softplus"]]
                     linear.output = TRUE,startweights = list(
                       matrix(winit[1:weights_input_to_hidden], nrow = n_input, ncol = n_hidden),
                       matrix(winit[(weights_input_to_hidden + 1):(weights_input_to_hidden + weights_hi
                              nrow = n_hidden, ncol = n_output)
                     ))
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_softplus,te), col="red", cex=1)

## comment your results

# task 3

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

new_predictions <- predict(nn, new_data)

plot(new_data$Var, new_data$Sin, col = "blue", main = "Neural Network Predictions on [0, 50]",
     xlab = "x", ylab = "sin(x)", cex = 1, pch = 16,ylim=c(-4,1.5))
points(new_data$Var, predict(nn, new_data), col = "red", cex = 1, pch = 16)
legend("topright", legend = c("True Values", "Predictions"),
       col = c("blue", "red"), pch = 16)


# task 4

weights <- nn$weights
print(weights)
```

```r
# comment

# task 5

Var_inv <- runif(500, 0, 10)
Sin_inver <- sin(Var_inv)
data_inver <- data.frame(Sin_inver, Var_inv)

nn_inver <- neuralnet(Sin_inver ~ Var_inv, data = data_inver, hidden = 10, act.fct = "logistic", linear
                startweights = list(
                  matrix(winit[1:weights_input_to_hidden], nrow = n_input, ncol = n_hidden),
                  matrix(winit[(weights_input_to_hidden + 1):(weights_input_to_hidden + weights_hidden_
                         nrow = n_hidden, ncol = n_output)
                ))
predictions_inver <- predict(nn_inver, data_inver)

plot(data_inver$Sin_inver, data_inver$Var_inv, col = "blue", main = "Neural Network Predictions: sin(x)
     xlab = "sin(x)", ylab = "x", cex = 1, pch = 16,ylim = c(-1,10))
points(data_inver$Sin_inver, predictions_inver, col = "red", cex = 1, pch = 16)
```