Assignment 3

Assignment 3 Neural Networks

1.

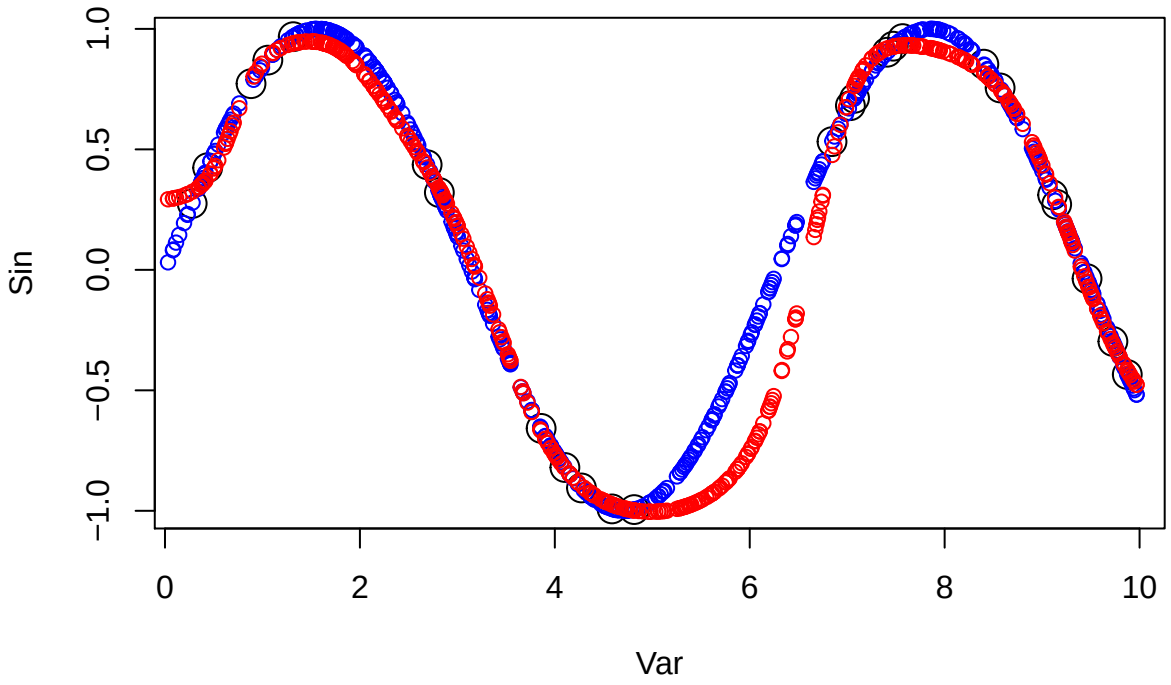
Train a neural network to learn the trigonometric sine function. To do so, sample 500 points uniformly at random in the interval [0,10].

```
Var <- runif(500, 0, 10) # sample 500 points uniformly at random in the interval [0;10]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
tr <- mydata[1:25,] # Use 25 of the 500 points for training
te <- mydata[26:500,] # Test


# Random initialization of the weights in the interval [-1, 1]
winit <- runif(25, -1, 1)# Your code here

# Use any number of layers and hidden units that you consider appropriate

nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit)
pred <- predict(nn,te)
#plot(nn)
#resError <- sum((te[,2]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2) # Plot the training
points(te, col = "blue", cex=1) # Plot test data
points(te[,1],predict(nn,te), col="red", cex=1) # plot predictions
```



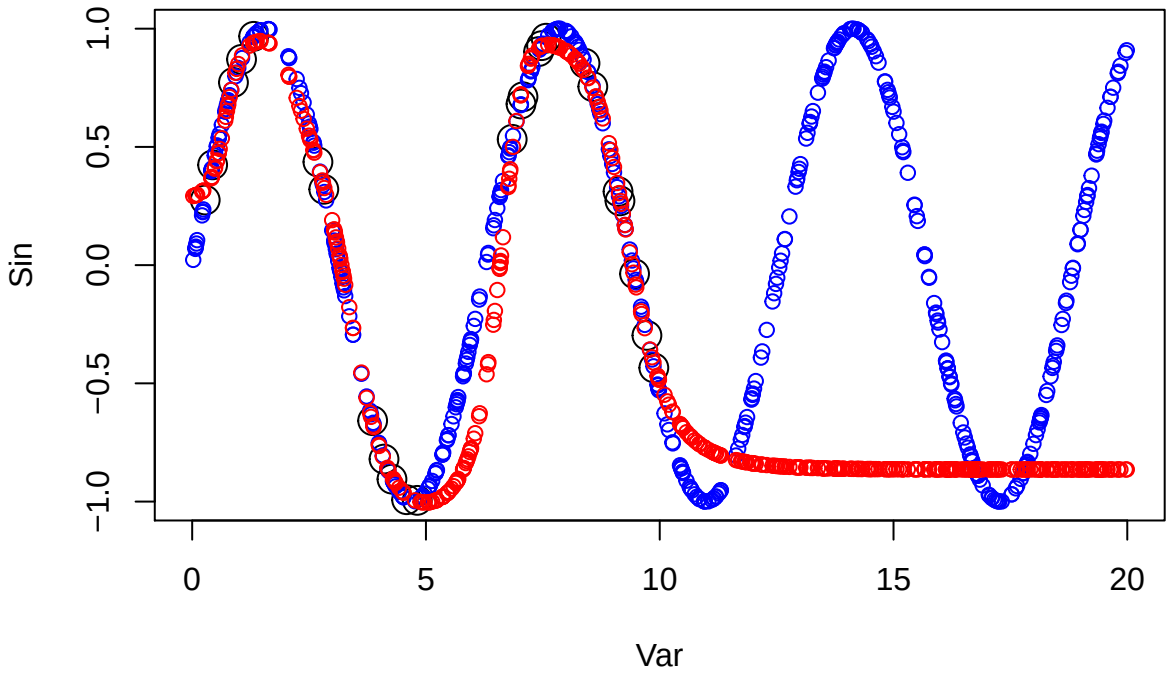

```
# Comment your results:
```

Above graph depicts that, the predicted points approximately interpolate the true values. However there are certain points where we could see some deviation, for example, the interval between [5,7.5]. This is due to very less training points.

2

Then, sample 500 points uniformly at random in the interval [0,20], and apply the sine function to each point. Use the previously learned NN to predict the sine function value for these new 500 points.

```
Var <- runif(500, 0, 20) # sample 500 points uniformly at random in the interval [0;20]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
pred <- predict(nn,mydata)
#resError <- sum((mydata[,2]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2,xlim=c(0,20),ylim=c(-1,1)) # Plot the training
points(mydata, col = "blue", cex=1) # Plot test data
points(mydata[,1],pred, col="red", cex=1) # plot predictions
```

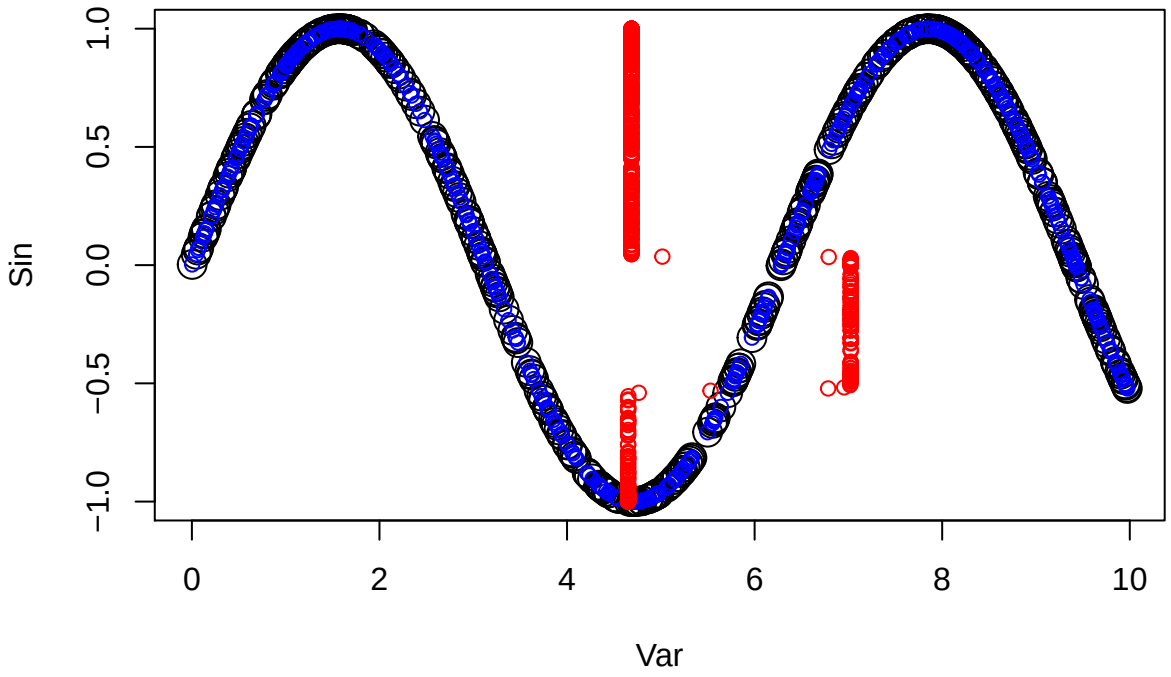


We have trained the model with the uniformly distributed points within the interval 0 and 10. Here, we are trying to fit the model with the data points that are from interval 0-20. Therefore, in the above graph, we could see that for the data points within 0-10 range the model fits well. And beyond that there are errors. That is model couldn't extrapolate those data points. Hence we see a mixed result.

3

Finally, sample 500 points uniformly at random in the interval [0,10], and apply the sine function to each point. Use all these points as training points for learning a NN that tries to predict x from sin(x), i.e. unlike before when the goals was to predict sin(x) from x. You should get bad results.

```
Var <- runif(500, 0, 10) # sample 500 points uniformly at random in the interval [0;10]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
nn <- neuralnet(formula = Var ~ Sin, data = mydata, hidden = 2, startweights = winit)
#plot(nn)
pred <- predict(nn,mydata)
#resError <- sum((mydata[1,]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(mydata, cex=2) # Plot the training
points(mydata, col = "blue", cex=1) # Plot test data
points(pred,mydata$Sin, col="red", cex=1) # plot predictions
```



In this case, we are trying to predict the x values given its *Sin* function. In the above graph Let us consider *Sin(0)*, *Sin(0)* has multiple (around 4) X values, neural network will take the average of those 4 X values and it is true with majority of data points. Hence we could see a straight line at the center of the plot.

In addition, there is another line at the right side, this is because, there are slightly more data points on the right side (for the interval [-0.5,0] on Y axis).

Finally, we could see some individual points in the interval $\sim$[5,7]. Because there are some gaps in the training points in that region.

The $\sin(x)$ is periodic in nature, neural network tries to average out the repeated values and converges with high error rates given less nodes (up to 3) and single hidden layer. Beyond this setting the model is not converging at all.