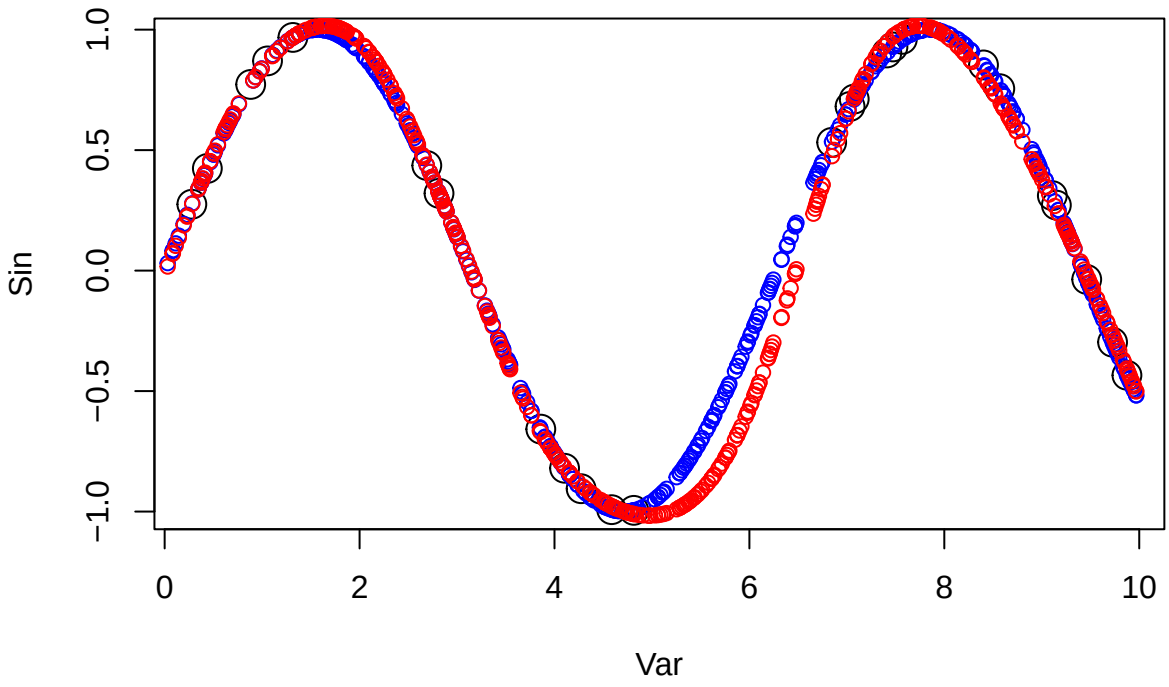
```
<style>
body {
text-align: justify}
</style>
```

Assignment 3

Task 1

- Train a neural network to learn the trigonometric sine function.
- Use 25 of the 500 points for training and the rest for test. Use one hidden layer with 10 hidden units. You do not need to apply early stopping. Plot the training and test data, and the predictions of the learned NN on the test data. You should get good results.
- Comment your results.

Overall, with the small amount of training data, the model fits the test data well. The interval between about 5 and 7 has a slightly worse fit. This might be due to the lack of data points in the training data for this range.



Task 2

- Repeat question (1) with the following custom activation functions: $h_1(x) = x$, $h_2(x) = max(0, x)$, $h_3(x) = ln(1 + e^x)$ (a.k.a. linear, ReLU and softplus). See the help file of the neural net package to learn how to use custom activation functions.

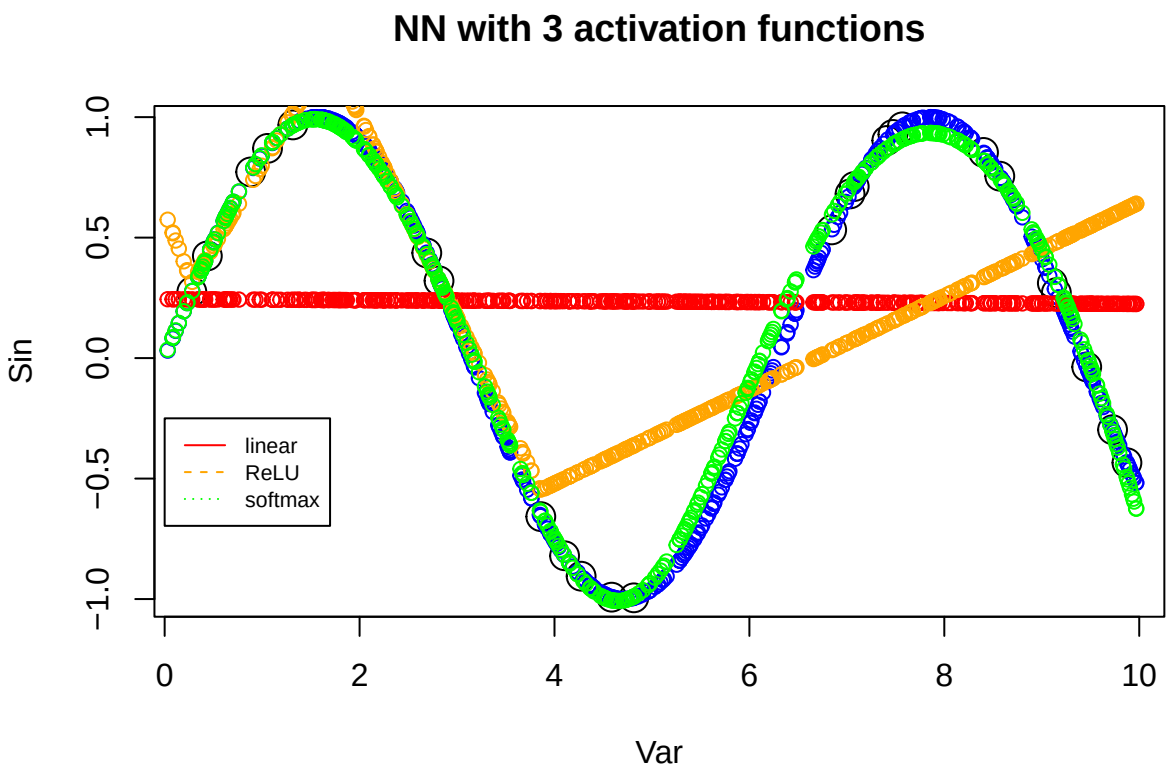
- Plot and comment your results.

The three activation functions have very different results.

The first, linear, has a near constant value of 0.2343153. This is because it is explicitly linear, which naturally will not fit a sine function very well. The reason it does not approximate 0 is because there are more data points in the first quadrant then the fourth.

The second, ReLU, is clearly linear in segments of the data. However, it has points for which it is non-linear, which is explained by the lack of a defined derivative of the max function. It does not predict the test data very well.

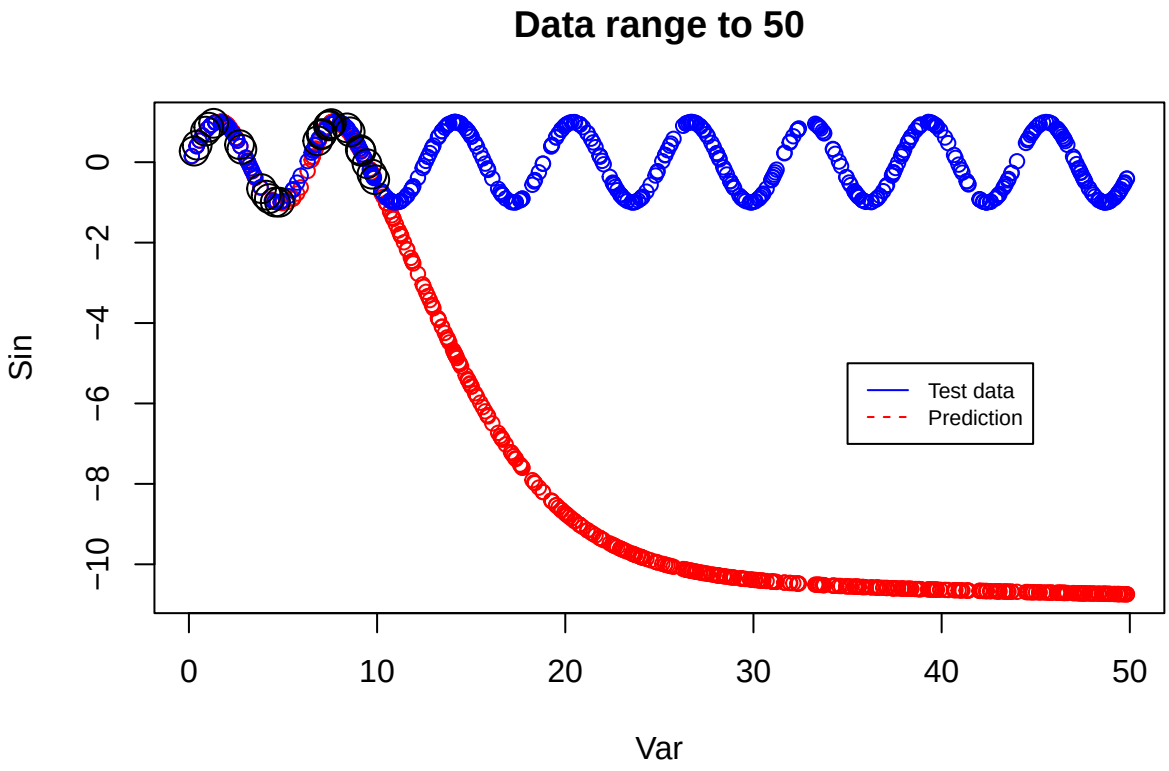
The third, softmax, approximates the test data well. It has infinitely many derivatives and arguably performs better than the sigmoid activation function in task 1, especially in the interval 5 to 7.



Task 3

- Sample 500 points uniformly at random in the interval [0,50], and apply the sine function to each point. Use the NN learned in question (1) to predict the sine function value for these new 500 points. You should get mixed results.
- Plot and comment on your results.

Observing the graph, the predictions from the first neural network does predict the data well on the interval of 0 to 10. This is because the model is trained on this range. However, beyond 10, the sigmoid activation function results in an extrapolated logistic curve.



Task 4

- In question (3), the predictions seem to converge to some value. Explain why this happens.

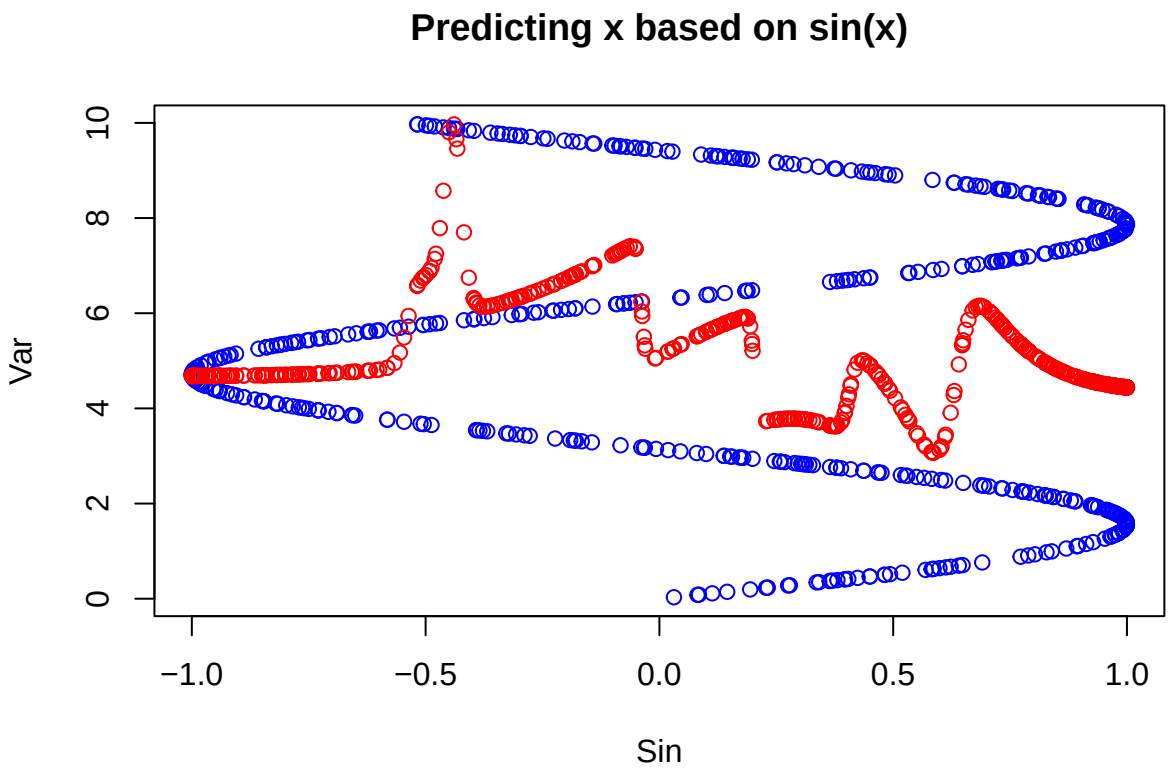
We did not choose a specific seed for the starting weights for the NN. One randomization gives -10.74471 as the minimum value which the prediction converges towards. As explained above, the reason the prediction deviates much more after Var = 10 is because it has not been trained on this data. Instead, it will be based on the derivatives from the sigmoid function

$$h(x) = \frac{1}{1 + e^{-x}}.$$

Task 5

- 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 goal was to predict sin(x) from x. Use the learned NN to predict the training data. You should get bad results.
- Plot and comment your results.

This is not a good fit. The main reason is because of the nature of the arcsine function: it has multiple results for a given x, unless the output is constrained. In this case, it is constrained between 0 and 10. But this includes multiple periods of the sine function. Therefore, the NN will have problems trying to predict multiple outcomes for a given input.



Assignment 1 Appendix

```
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
# 31 randomized weights with 20 for hidden nodes and 11 for outcome node
# This is evident when plotting the network
winit = runif(31,-1,1)

# Create a neural network model with 1 hidden layer and 10 hidden units
nn = neuralnet(Sin~., tr, hidden = 10, startweights = winit)
#plot(nn)
# 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)
linear = function(x) x
ReLU = function(x) ifelse(x>0,x,0)
softmax = function(x) log(1 + exp(x))

# Create a neural network model with 1 hidden layer and 10 hidden units
# with 3 different activation functions
nn_r = neuralnet(
```

```
  Sin~., tr, hidden = 10, act.fct = ReLU, startweights = winit, threshold = 0.1)
pred_r = predict(nn_r,te)

nn_l = neuralnet(
  Sin~., tr, hidden = 10, act.fct = linear, startweights = winit)
pred_l = predict(nn_l,te)
#mean(pred_l)

nn_s = neuralnet(
  Sin~., tr, hidden = 10, act.fct = softmax, startweights = winit)
pred_s = predict(nn_s,te)

plot(tr, cex=2, main = "NN with 3 activation functions")
points(te, col = "blue", cex=1)
points(te[,1],pred_l, col="red", cex=1)
points(te[,1],pred_r, col="orange", cex=1)
points(te[,1],pred_s, col="green", cex=1)
legend(
  0,-0.25, legend = c("linear", "ReLU","softmax"),
  col = c("red", "orange", "green"), cex = 0.7, lty = 1:3)
set.seed(1234567890)
Var = runif(500,0,50)
mydata_2 = data.frame(Var, Sin = sin(Var))
preds = predict(nn, mydata_2)
#min(preds)

plot(
  mydata_2[,1], preds, col="red", cex=1, xlab = "Var",
  ylab = "Sin", main = "Data range to 50")
points(mydata_2[,1], mydata_2[,2],  col = "blue", cex=1)
points(tr, cex = 2)
legend(
  35,-5, legend = c("Test data", "Prediction"),
  col = c("blue", "red"), lty = 1:2, cex = 0.7)
plot(nn)
set.seed(1234567890)
Var = runif(500,0,10)
mydata_3 = data.frame(Var, Sin = sin(Var))

nn_2 = neuralnet(
  Var~., mydata_3, hidden = 10, startweights = winit,
  threshold = 0.1)

preds_2 = predict(nn_2, mydata_3)

plot(
  mydata_3[,2], mydata_3[,1], col = "blue", cex=1,
  xlab = "Sin", ylab = "Var", main = "Predicting x based on sin(x)")
points(mydata_3[,2], preds_2, col="red", cex=1)
```