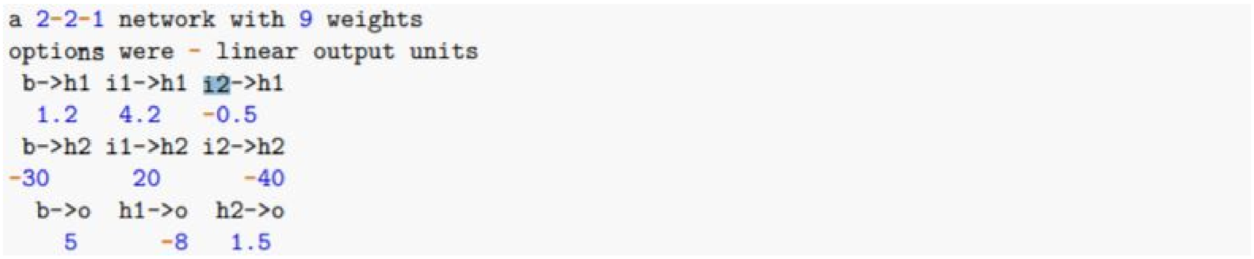
Statistical Learning HW 3 - Neural Networks

Christian Conroy

March 1, 2019

Extra 28 (3 Points)

Below is the output from nnet after we fit a model. Let's assume we used a tanh() activation function throughout. Let xi, i = 1, 2, . . . be the input variables and let h1, h2, . . . be the output from the hidden layer.



```
a 2-2-1 network with 9 weights
options were - linear output units
 b->h1 i1->h1 i2->h1
  1.2   4.2   -0.5
 b->h2 i1->h2 i2->h2
-30      20      -40
  b->o  h1->o  h2->o
    5      -8   1.5
```

Figure 1: Alt text

(a) Draw a diagram of this neural network architecture. Label all the edges with the corresponding weights.

(b) Provide an expression for the output value of the first hidden unit as a function of the values of the input features. This should have the form h1 = f(x1, x2, . . .) for a suitable explicit function f.

h1 = f(1.2b + 4.2x1 -0.5x2)

(c) Provide an expression for the value at the output node as a function of the values at the hidden units. This should have the form z = g(h1, h2, . . .) for a suitable explicit function g.

z = g(5b - 8h1 + 1.5h2)

(d) Provide an expression for the value at the output node as a function of the input values. This should have the form z = F(x1, x2, . . .) for a suitable explicit function F.

z = F(5b -8(1.2b + 4.2x1 -0.5x2) + 1.5(-30b +20x1 - 40x2))

Extra 30 (3 points)

Suppose two different ANNs have been trained on a training set for a classification problem, and the responses, scaled to have values in [0, 1], have been computed for all training instances for both networks. Assume that the responses for network 2 are related to those from network 1 by a monotone function, such as in the plot below. Explain carefully why the two ANNs have the same ROC curve.

Given that we are training both networks on the same train set, the sensitivity (True Positive) and 1 - specificity (False Positive) axes that comprise the ROC curve at different decision thresholds (the probability threshold considered to assign the positive class) will be the same for both networks. If the responses are related by a monotone function abd the responses from the train set are scaled [0,1], then we are essentially dealing with an ANN with a logistic activiation function, or a logistic regression model.

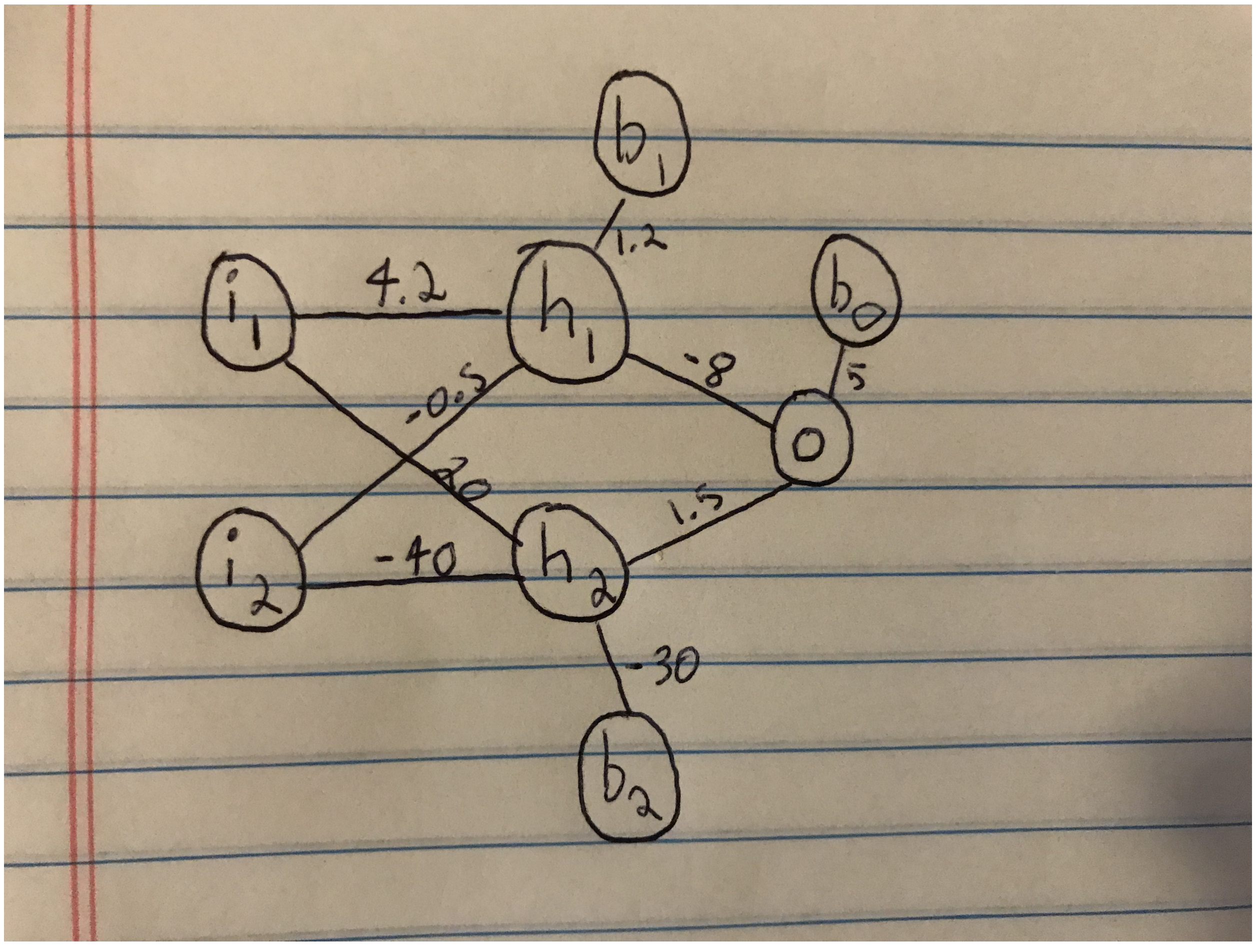


Figure 2: Alt text

Extra 34 (3 points)

Make a dataframe with k = 11 columns and N = 100 observations, where all entries are independent standard normal random sample. Let z be the last column. Use set.seed(20305).

```r
set.seed(20305)
data <- data.frame(matrix(rnorm(100), nrow = 100, ncol = 11, byrow = TRUE))
colnames(data)[11] <- "z"
```

(a) Fit z to the other 10 columns using multiple regression. What is the sum of squares of the residuals?

```r
reg1 <- lm(z ~ ., data = data)
summary(reg1)
```

```
##
## Call:
## lm(formula = z ~ ., data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.168 -0.546  0.039  0.549  2.645
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1300     0.1084    1.20     0.23
## X1           -0.0284     0.1060   -0.27     0.79
## X2            0.0553     0.1060    0.52     0.60
## X3           -0.0834     0.1058   -0.79     0.43
## X4           -0.0886     0.1057   -0.84     0.40
## X5           -0.0663     0.1055   -0.63     0.53
## X6           -0.1215     0.1055   -1.15     0.25
## X7            0.0362     0.1057    0.34     0.73
## X8           -0.0708     0.1058   -0.67     0.50
## X9            0.0631     0.1060    0.60     0.55
## X10          -0.0598     0.1060   -0.56     0.57
##
## Residual standard error: 1.02 on 89 degrees of freedom
## Multiple R-squared:  0.0535, Adjusted R-squared:  -0.0528
## F-statistic: 0.503 on 10 and 89 DF,  p-value: 0.884
```

```r
anova(reg1)
```

```
## Analysis of Variance Table
##
## Response: z
##           Df Sum Sq Mean Sq F value Pr(>F)
## X1         1    0.0   0.048    0.05   0.83
## X2         1    0.2   0.233    0.22   0.64
## X3         1    0.4   0.375    0.36   0.55
## X4         1    1.1   1.081    1.04   0.31
## X5         1    0.2   0.171    0.16   0.69
## X6         1    1.8   1.773    1.70   0.20
## X7         1    0.2   0.227    0.22   0.64
## X8         1    0.6   0.589    0.57   0.45
## X9         1    0.4   0.416    0.40   0.53
## X10        1    0.3   0.332    0.32   0.57
## Residuals 89   92.8   1.042
```

The sum of squared errors is 92.8.

(b) Fit z to the other 10 columns, using a neural network with two hidden units and setting maxit = 2000 and decay = .01. Does this model fit the data better? How do you know?

```
nn_fit <- nnet(z ~ ., data=data, maxit = 2000, decay = .01,  size=2)
```

```
## # weights:  25
## initial  value 103.519040
## iter  10 value 98.170869
## iter  20 value 86.697084
## iter  30 value 83.853586
## iter  40 value 83.076781
## iter  50 value 83.032330
## iter  60 value 83.021933
## final  value 83.021895
## converged
```

This model fits the data better because we've added more hidden nodes, thereby making our model more flexible. The data converges (i.e. no longer fitting or doing gradient descent) at 83.02 (cross-entropy here is comparable to the SSR above), meaning we have a lower loss function than we did in the regression model (SSR of 92.8)

(c) Redo this experiment with the same data and with 5 and 10 hidden units and explain what you see.

```
nn_fit <- nnet(z ~ ., data=data, maxit = 2000, decay = .01,  size=5)
```

```
## # weights:  61
## initial  value 114.824303
## iter  10 value 99.389701
## iter  20 value 89.005620
## iter  30 value 84.418027
## iter  40 value 83.957663
## iter  50 value 79.743775
## iter  60 value 75.989300
## iter  70 value 75.301282
## iter  80 value 71.536431
## iter  90 value 64.858485
## iter 100 value 63.042621
## iter 110 value 62.147478
## iter 120 value 61.886012
## iter 130 value 61.764596
## iter 140 value 61.720205
## iter 150 value 61.713558
## iter 160 value 61.712330
## iter 170 value 61.712165
## final  value 61.712161
## converged
```

As expected, increasing the number of hidden nodes from 2 to 5 decreases the loss function even further to 63.97.

```
nn_fit <- nnet(z ~ ., data=data, maxit = 2000, decay = .01,  size=10)
```

```
## # weights:  121
## initial  value 101.036142
## iter  10 value 96.993502
## iter  20 value 84.799456
## iter  30 value 83.746893
## iter  40 value 82.419975
## iter  50 value 81.107861
## iter  60 value 76.285676
## iter  70 value 75.588551
## iter  80 value 72.379225
## iter  90 value 63.615488
## iter 100 value 60.518175
## iter 110 value 60.013365
## iter 120 value 59.720134
## iter 130 value 59.416294
## iter 140 value 59.255925
## iter 150 value 59.211544
## iter 160 value 59.193762
## iter 170 value 59.116708
## iter 180 value 59.040275
## iter 190 value 59.016411
## iter 200 value 58.988584
## iter 210 value 58.980605
## iter 220 value 58.979042
## iter 230 value 58.978856
## iter 240 value 58.978837
## iter 240 value 58.978837
## iter 240 value 58.978837
## final  value 58.978837
## converged
```

As expected, increasing the number of hidden nodes even further from 5 to 10 decreases the loss function even further to 58.35.

Extra 33 (5 points)

We'll use the MNIST image classification data, available as mnist_all.RData that were used in class during the last two weeks. We want to distinguish between 4 and 7. Extract the relevant training data and place them in a data frame.

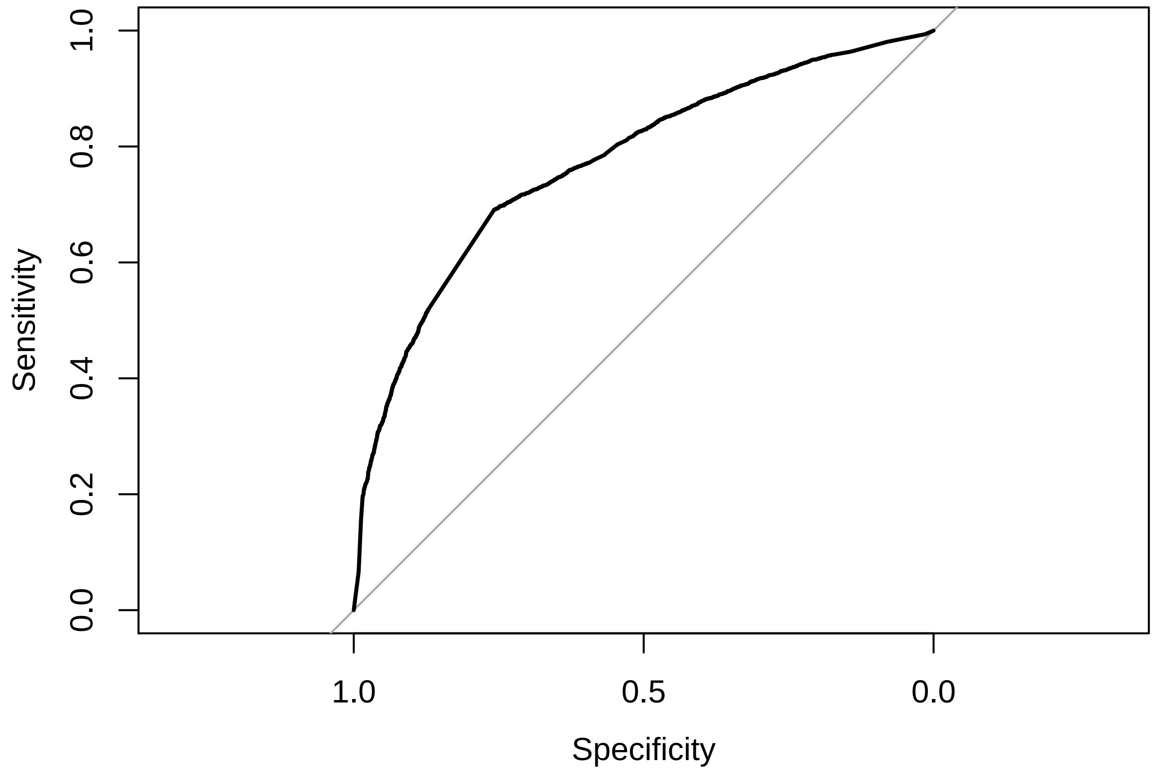
```r
data <-load('mnist_all.RData')
train <- data.frame(train$n, train$x, train$y)
train <- train[train$train.y == 4 | train$train.y == 7,]
test <- data.frame(test$n, test$x, test$y)
test <- test[test$test.y == 4 | test$test.y == 7,]
```

(a) Pick two features (variables) that have large variances and low correlation. Fit a logistic regression model with these two features. Evaluate the model with the AUC score.

```r
lvlc <- sort(sapply(train , function(x) sd(x)), decreasing = TRUE)[1:20]
lvlc <- train[,names(lvlc)]
cor(lvlc)

# X346 and x602 have high variance and low correlation.
```

```r
spec <- as.formula("factor(train.y) ~ X346 + X602")
```

```r
fit <- glm(formula = spec, data = train, family=binomial)

train$pred <- predict(fit, type = 'response')

r <- roc(train$train.y, train$pred)
plot(r)
```



```r
auc(train$train.y, train$pred)
```

```
## Area under the curve: 0.772
```

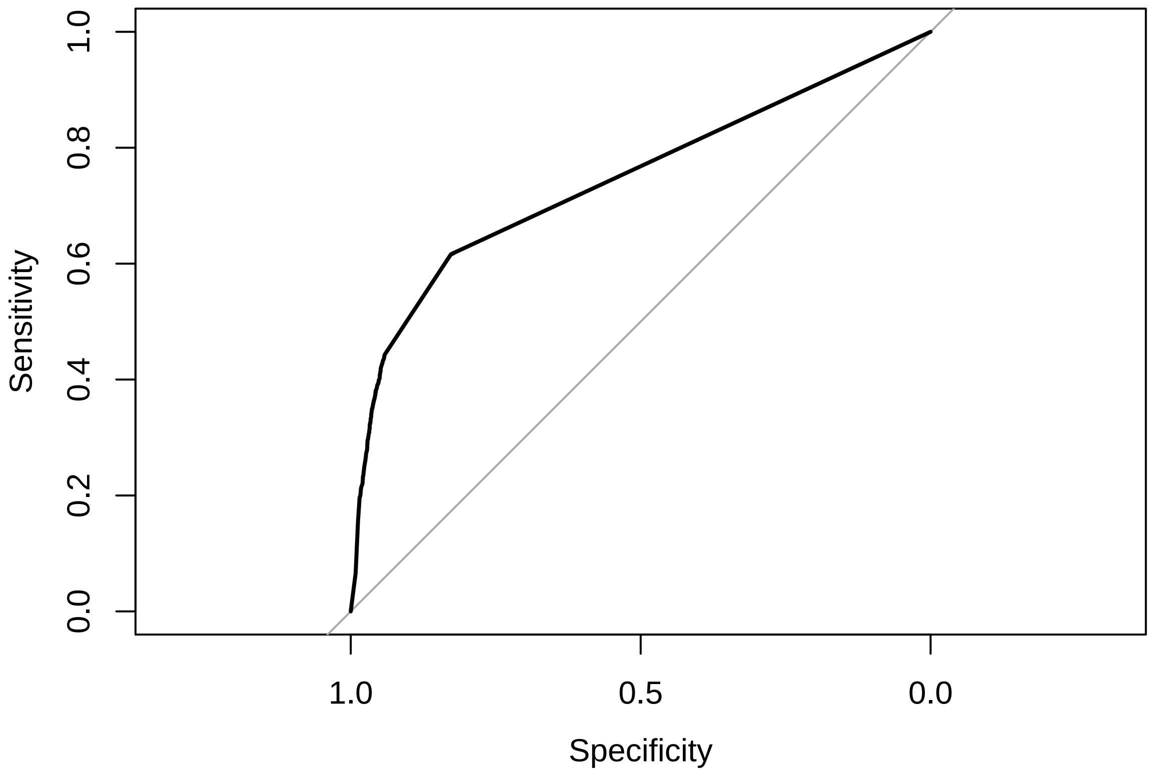
The AUC is 0.772, meaning that it performns alright but not great.

(b) Create a neural net with one unit in the hidden layer. Train the neural net with the same two features as the previous part and evaluate the model with AUC. Compare to the results from (a) and explain.

```r
nn_fit <- nnet(formula = spec, data=train, maxit = 2000, decay = .01,  size=1)
```

```
## # weights:  5
## initial  value 8904.847464
## iter  10 value 6945.201478
## iter  20 value 6857.829755
## iter  30 value 6846.002394
## iter  40 value 6838.127984
## iter  50 value 6837.903772
## final  value 6837.454464
## converged
```

```r
train$pred_nn <- predict(nn_fit, type = "r")

r <- roc(train$train.y, train$pred_nn)
plot(r)
```



```r
auc(train$train.y, train$pred_nn)
```

```
## Area under the curve: 0.745
```

The neural network model with one hidden layer actually does worse than the logistic model.

(c) With the same two features, train three different neural nets, each time using more units in the hidden layer. How do the results improve, using the AUC?

```r
nn_fit2 <- nnet(formula = spec, data=train, maxit = 2000, decay = .01,  size=2)
```

```
## # weights:  9
## initial  value 8607.652293
## iter  10 value 6787.220271
## iter  20 value 6744.200230
## iter  30 value 6659.992150
## iter  40 value 6652.132648
## iter  50 value 6651.295953
## iter  60 value 6651.034465
## iter  70 value 6650.984180
## final  value 6650.982669
## converged
```

```
train$pred_nn2 <- predict(nn_fit2, type = "r")

auc(train$train.y, train$pred_nn2)
```

```
## Area under the curve: 0.782
```

```
nn_fit5 <- nnet(formula = spec, data=train, maxit = 2000, decay = .01,  size=5)
```

```
## # weights:  21
## initial  value 8551.820658
## iter  10 value 6855.649044
## iter  20 value 6693.498410
## iter  30 value 6641.121155
## iter  40 value 6634.420139
## iter  50 value 6632.113541
## iter  60 value 6622.491940
## iter  70 value 6611.453552
## iter  80 value 6608.768684
## iter  90 value 6607.941568
## final  value 6607.793969
## converged
```

```
train$pred_nn5 <- predict(nn_fit5, type = "r")

auc(train$train.y, train$pred_nn5)
```

```
## Area under the curve: 0.788
```

```
nn_fit10 <- nnet(formula = spec, data=train, maxit = 2000, decay = .01,  size=10)
```

```
## # weights:  41
## initial  value 10175.176131
## iter  10 value 6724.726066
## iter  20 value 6646.309963
## iter  30 value 6631.225676
## iter  40 value 6617.736399
## iter  50 value 6611.523539
## iter  60 value 6609.441181
## iter  70 value 6605.720802
## iter  80 value 6605.308005
## iter  90 value 6604.615895
## iter 100 value 6603.922831
## iter 110 value 6603.682384
## iter 120 value 6603.485329
## iter 130 value 6603.453476
## iter 140 value 6603.450232
## final  value 6603.449303
## converged
```

```
train$pred_nn10 <- predict(nn_fit10, type = "r")

auc(train$train.y, train$pred_nn10)
```

```
## Area under the curve: 0.79
```

The neural network model significantly improves the AUC from the from the model with the logistic regression to the ANN with two hidden layers to the ANN with five hidden layers. There is an improvement from the nnet model with five hidden layers to the model with ten hidden layers as well. It is interesting that there is not an improvement from the logstic model to the nnet models with one layer.

(d) Is there evidence for overfitting in your results in (c)? Use the test data, also availabe in mnist_all.RData, to find out.

```r
test$pred_nn2 <- predict(nn_fit2, newdata = test, type = "r")

auc(test$test.y, test$pred_nn2)
```

```
## Warning in roc.default(response, predictor, auc = TRUE, ...): Deprecated
## use a matrix as predictor. Unexpected results may be produced, please pass
## a numeric vector.

## Area under the curve: 0.787
```

```r
test$pred_nn5 <- predict(nn_fit5, newdata = test, type = "r")

auc(test$test.y, test$pred_nn5)
```

```
## Warning in roc.default(response, predictor, auc = TRUE, ...): Deprecated
## use a matrix as predictor. Unexpected results may be produced, please pass
## a numeric vector.

## Area under the curve: 0.791
```

```r
test$pred_nn10 <- predict(nn_fit10, newdata = test, type = "r")

auc(test$test.y, test$pred_nn10)
```

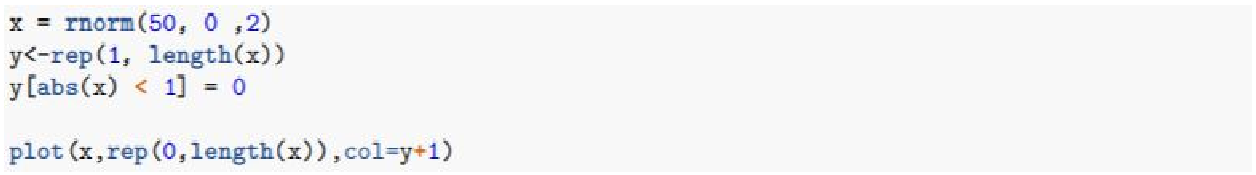
```
## Warning in roc.default(response, predictor, auc = TRUE, ...): Deprecated
## use a matrix as predictor. Unexpected results may be produced, please pass
## a numeric vector.

## Area under the curve: 0.793
```

The auc for the test set is not drastically different from that of the train set. For the hidden node with two layers, the test set does only slightly worse. The auc for the test set actually does better for five nodes and ten nodes. There is therefore not significant evidence for overfitting. As indicated by the AUCs on the train sets, the bias was not that low, so we should not expect that high of a variance.

5 points each:

Extra 36 (5 points)

In the Tensorflow Playground, we can use a "bullseye" dataset to demonstrate non-linear decision boundaries that would be impossibly difficult for logistic regression. Here we're going to explore that kind of dataset in a simplified version.A one-dimensional bullseye dataset would be like the following. Notice that one of the classes completely surrounds the other.

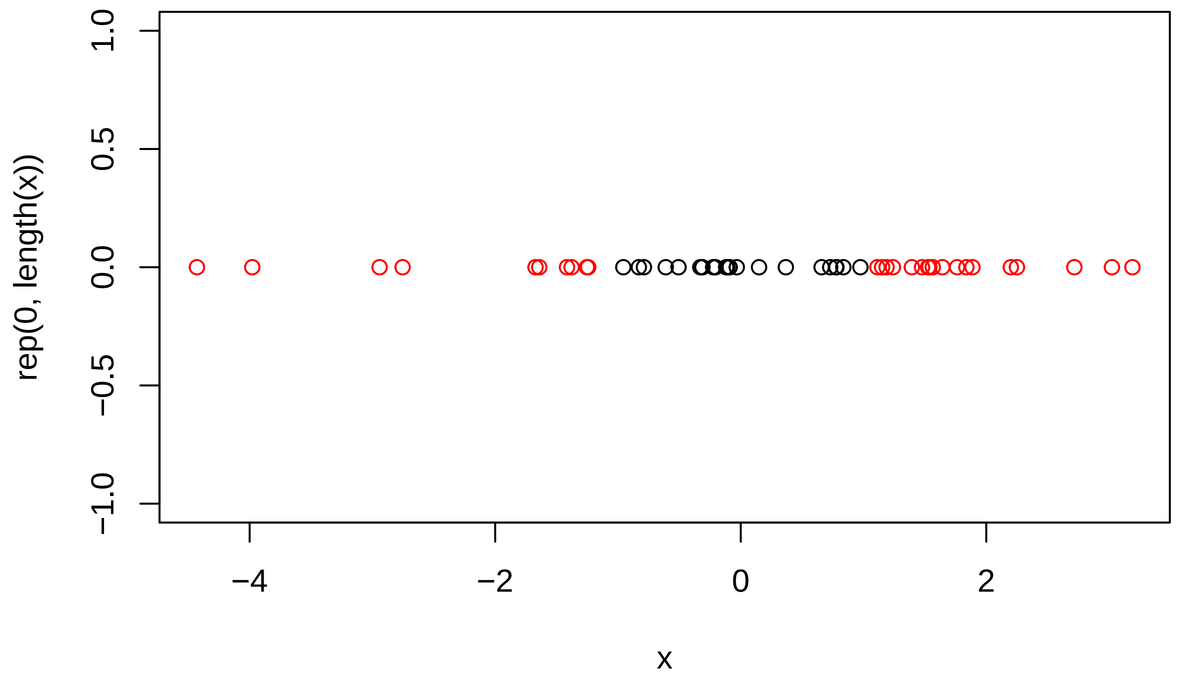


```r
x = rnorm(50, 0 ,2)
y<-rep(1, length(x))
y[abs(x) < 1] = 0

plot(x,rep(0,length(x)),col=y+1)
```

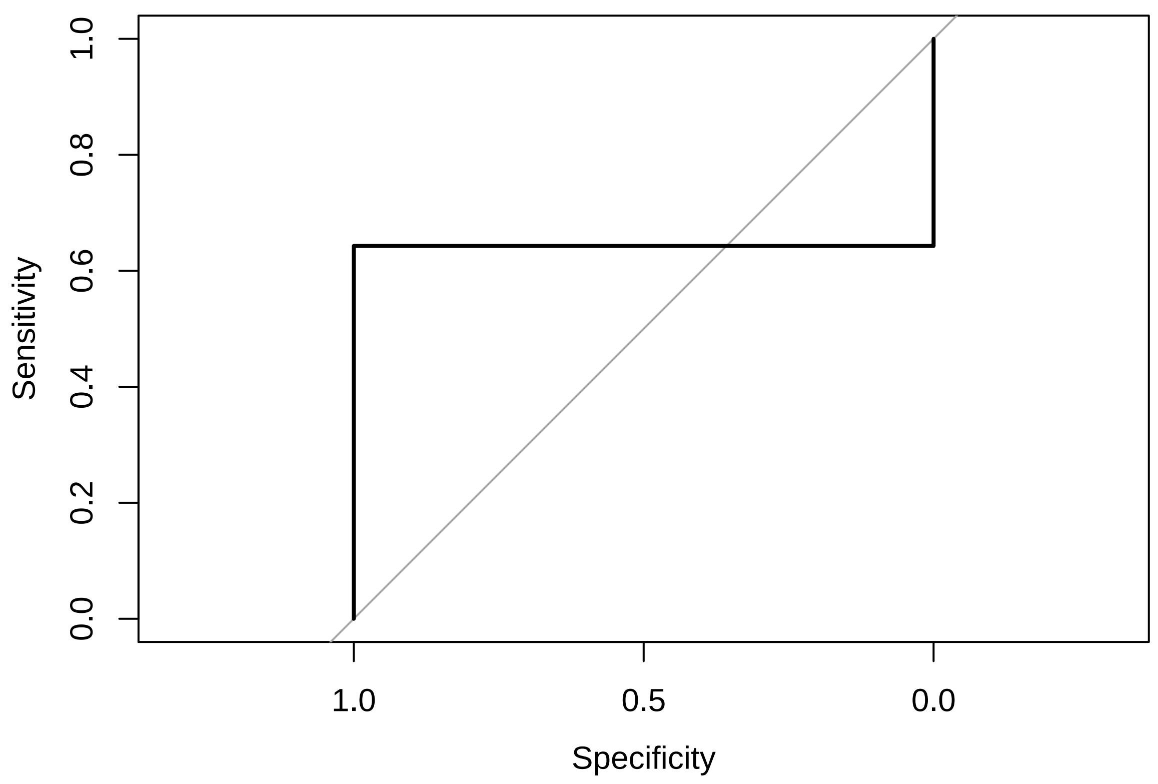
Figure 3: Alt text

(a) Fit a logistic regression model to this dataset. Verify that the results are not great.

```
set.seed(1)
x = rnorm(50, 0 ,2)
y<-rep(1, length(x))
y[abs(x) < 1] = 0
plot(x,rep(0,length(x)),col=y+1)
```



```
data <- data.frame(x, y)

fit <- glm(y ~ x, data = data, family=binomial)

data$pred <- predict(fit, type = 'response')

r <- roc(data$y, data$pred)
plot(r)
```



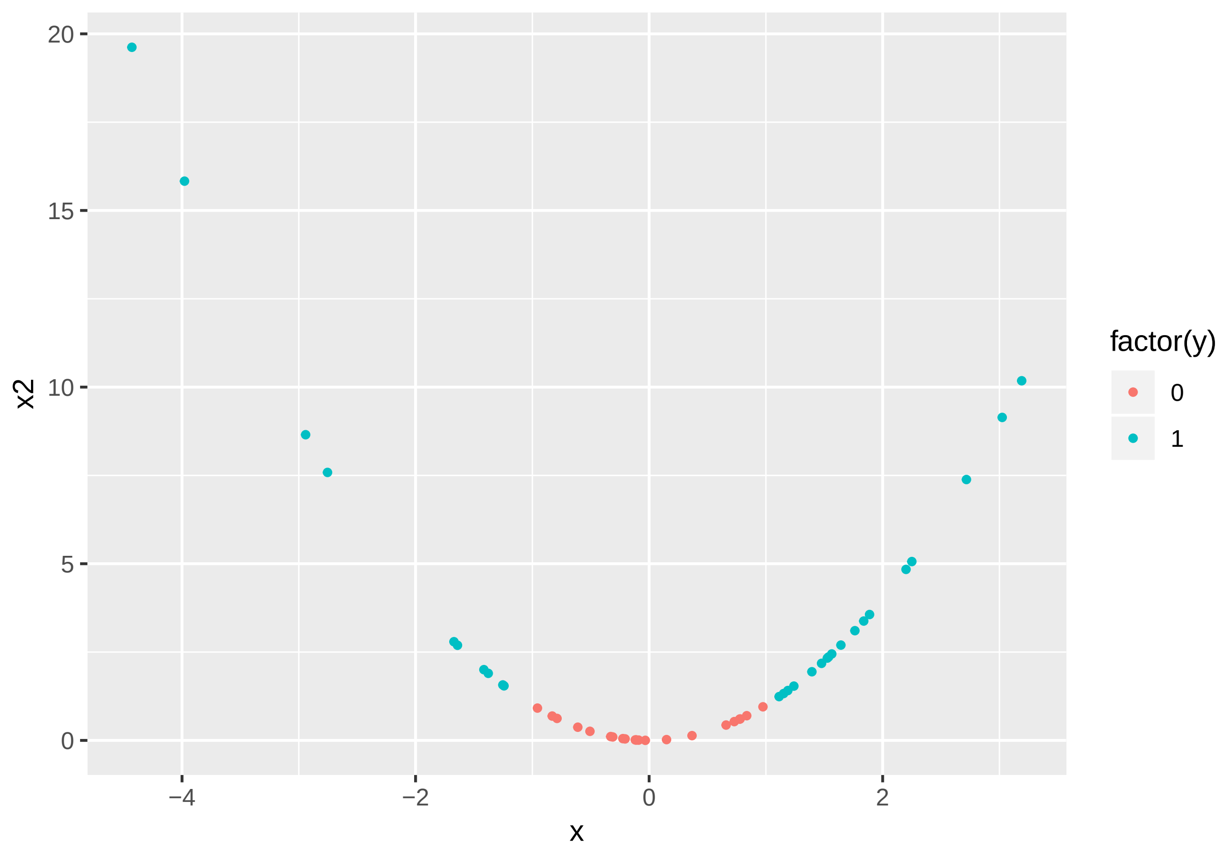
```
auc(data$y, data$pred)
```

```
## Area under the curve: 0.643
```

The AUC of about .643 is close to the point at .50 where just switching all of our classifications would improve the classification. The ROC curve also shows clearly that the results are not great.

(b) But we can solve this problem using logistic regression if we employ clever "feature engineering". Create a new feature which is just x2. Make a plot of the two features x and x2 and color by class label to verify that the two classes are now more easily separable. Fit a logistic regression model and comment on the results.

```
data$x2 <- x^2
ggplot(data, aes(x = x, y = x2, color = factor(y))) + geom_point(size = 1, alpha = 1, na.rm = TRUE)
```

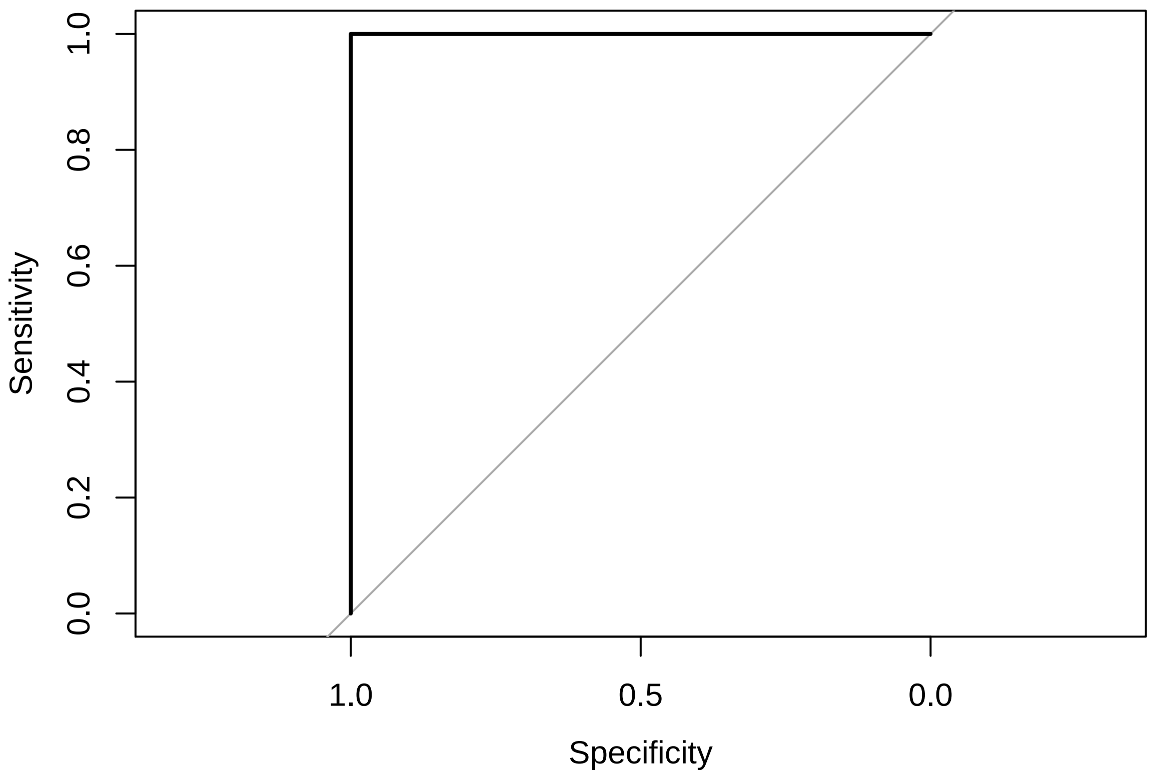


```r
fit <- glm(y ~ x + x2, data = data, family=binomial)
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ x + x2, family = binomial, data = data)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -1.37e-04  -2.00e-08   2.00e-08   2.00e-08   1.34e-04
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -147.3    39061.7       0        1
## x               13.7    25374.6       0        1
## x2             121.4    30656.2       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.8593e+01  on 49  degrees of freedom
## Residual deviance: 3.6937e-08  on 47  degrees of freedom
## AIC: 6
##
## Number of Fisher Scoring iterations: 25
```

```r
#
data$pred_2 <- predict(fit, type = 'response')
```

```r
#
r <- roc(data$y, data$pred_2)
plot(r)
```



```r
auc(data$y, data$pred_2)
```

```
## Area under the curve: 1
```

According to the ROC and AUC, this model predicts classifications perfectly. Some observations ended up with predicted probabilities of 1. It is important to note that neither X nor X2 are significant. In fact, they are highly insignificant, meaning that we've ended up with very high variability and very low bias (i.e. overfitting).

(c) If we never thought of this feature engineering, we can also easily solve this problem with a neural network. But importantly, we have to make a network topology such that the hidden layer has higher dimensionality than the input layer. Fit a neural network to Y ??? X with two nodes in the hidden layer. Verify that we can achieve perfect classification on the training data.

```r
nn_fit <- nnet(y ~ x, data= data, maxit = 2000, decay = .01,  size=2)
```

```
## # weights:  7
## initial  value 13.398152
## iter  10 value 8.893467
## iter  20 value 8.637542
## iter  30 value 5.236470
## iter  40 value 3.408634
## iter  50 value 3.357195
## final  value 3.357193
## converged
```

```r
data$pred_nn <- predict(nn_fit, type = "r")

auc(data$y, data$pred_nn)
```

```
## Area under the curve: 1
```

We achieve perfect classification on the data through this method.

(d) By projecting the data into a higher-dimensional space, we can separate the two classes. In the case of a neural network, the network figured it out for us - we didn't have to do it ourselves. Provide an explanation and intuition into how the network can achieve this goal in this particular case. Your explanation might rely on helpful visualizations.

When the two classification classes are linearly separable, then we can just use a logistic regression, which uses hyperplanes as decision boundaries. However, when we have nonlinear decision boundaries like we do in this case, neural networks are a useful tool because it uses matrix multiplication to identify a matrix of weights that minimizes the squared loss, as opposed to maximizing likelihood, through nonlinear functions. Any continuous function of n variables can be approximated arbitrarily well by a feed-forward artificial neural network with one hidden layer and finitely many laters (two nodes in that layer in this case) according to the universal approximation property of neural networks. Our model is therefore

Extra 37 (5 points)

The data for this exercise are in the UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/index.php. We shall use the concrete compressive strength data. For details and a link, refer to problem 11.

a) Import the data into your R workspace and change all variable names to something simpler. Split the data into a training set (70%) and a test set (30%).

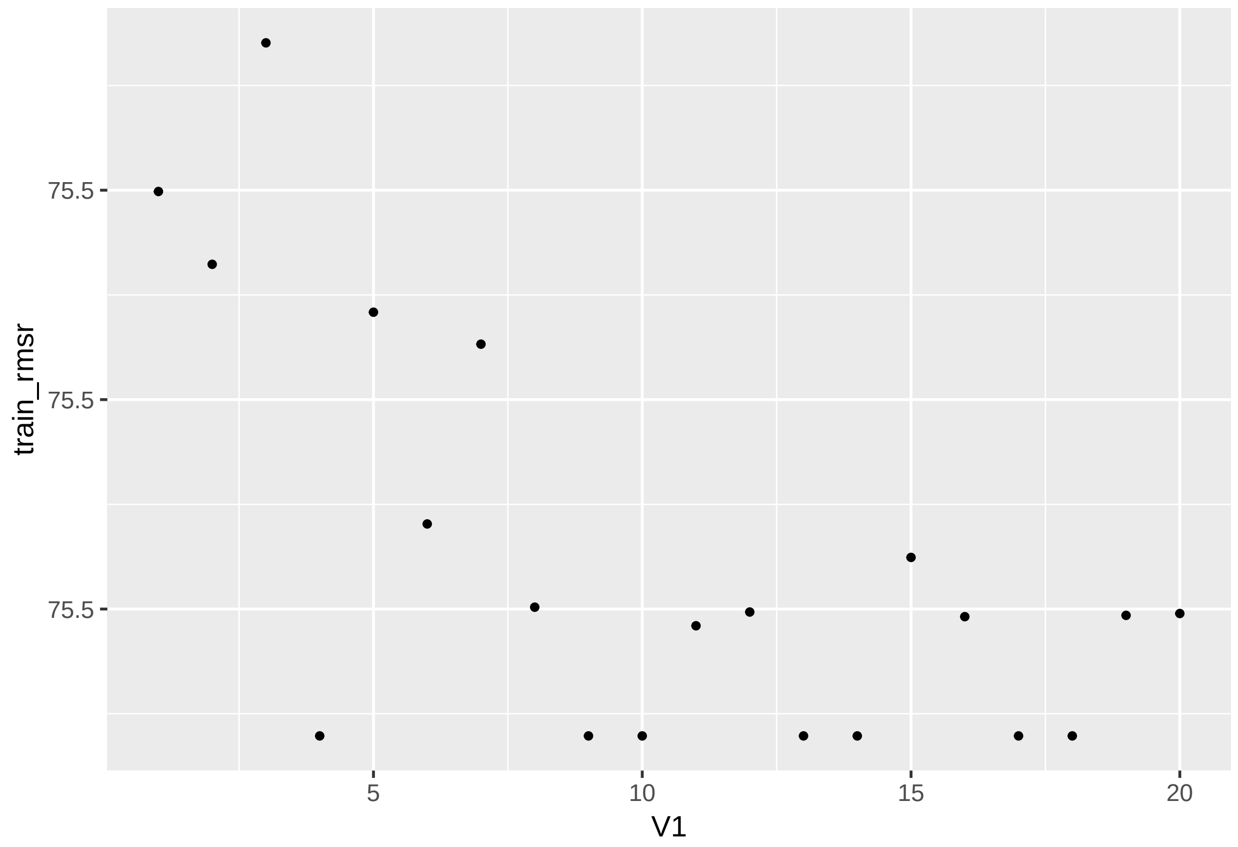
```r
concrete <- read_excel("Concrete_Data.xls")
colnames(concrete) <- c("cementkg", "blustfur", "flyash", "superplas", "courseagg", "fineagg", "age", "

# set the seed to make your partition reproductible
set.seed(246)
smp_size <- floor(0.80 * nrow(concrete))
train <- sample(seq_len(nrow(concrete)), size = smp_size)
concrete_train <- concrete[train,]
concrete_test <- concrete[-train,]
```

b) Fit artificial neural networks with a single hidden layer and 2, 3, 4, . . . 20 nodes to the training data. Compute the root mean squared residuals for each network and plot this quantity against the number of hidden nodes.

```r
train_rmsr <- numeric()

for(i in 1:20) {
  nn_fit <- nnet(CCS ~ ., data= concrete_train, maxit = 2000, decay = .01,  size = i)
  train_rmsr[i] <- sqrt(mean((predict(nn_fit, type = "r") - concrete_train$CCS)^2))
}
```

```
## # weights:  11
## initial  value 4719605.846369
## iter  10 value 4693761.066038
## final  value 4693759.090905
## converged
## # weights:  21
## initial  value 4720077.112824
## iter  10 value 4693761.572103
## final  value 4693759.038919
## converged
## # weights:  31
## initial  value 4740912.271542
## iter  10 value 4693789.989285
## iter  20 value 4693760.163129
## final  value 4693759.305674
## converged
## # weights:  41
## initial  value 4727684.839965
## iter  10 value 4694207.975873
## iter  20 value 4693774.087426
## iter  30 value 4693759.097906
## final  value 4693758.936661
## converged
## # weights:  51
## initial  value 4747977.363951
## iter  10 value 4693784.931301
## iter  20 value 4693759.213916
## final  value 4693758.813183
## converged
## # weights:  61
## initial  value 4703543.097364
## iter  10 value 4693759.666783
## final  value 4693758.569362
## converged
## # weights:  71
## initial  value 4730954.905412
## iter  10 value 4693759.506057
## iter  10 value 4693759.479089
## final  value 4693759.035354
## converged
## # weights:  81
## initial  value 4730742.229266
## iter  10 value 4693761.039890
## final  value 4693758.749623
## converged
## # weights:  91
## initial  value 4744621.415970
## iter  10 value 4694964.487179
## iter  20 value 4693806.522970
## iter  30 value 4693759.188802
## iter  30 value 4693759.148989
## iter  30 value 4693759.102746
## final  value 4693759.102746
## converged
## # weights:  101
## initial  value 4750616.115699
## iter  10 value 4693766.664089
## iter  20 value 4693758.931975
## final  value 4693758.377477
## converged
## # weights:  111
## initial  value 4716308.812495
## iter  10 value 4693931.692275
## final  value 4693758.655269
## converged
## # weights:  121
## initial  value 4715390.309844
## iter  10 value 4693765.900488
## final  value 4693759.458032
## converged
## # weights:  131
## initial  value 4712503.135523
## iter  10 value 4693767.928268
## final  value 4693758.395214
## converged
## # weights:  141
## initial  value 4744615.934732
## iter  10 value 4695365.125791
## iter  20 value 4693795.850849
## iter  30 value 4693758.448772
## final  value 4693758.253790
## converged
## # weights:  151
## initial  value 4705696.826160
## iter  10 value 4696051.134312
## iter  20 value 4694107.576509
## final  value 4693758.352035
## converged
## # weights:  161
## initial  value 4760700.179369
## iter  10 value 4693775.189426
## iter  20 value 4693758.513553
## final  value 4693758.320165
## converged
## # weights:  171
## initial  value 4743111.257917
## iter  10 value 4695257.462941
## iter  20 value 4693810.594291
## final  value 4693758.928702
## converged
## # weights:  181
## initial  value 4724052.983660
## iter  10 value 4693759.722584
## final  value 4693758.329378
## converged
## # weights:  191
## initial  value 4736266.299586
## iter  10 value 4695550.374578
## iter  20 value 4693775.252083
## iter  30 value 4693760.421962
## final  value 4693758.371889
## converged
## # weights:  201
## initial  value 4700409.238789
## iter  10 value 4693767.726971
## iter  20 value 4693758.273062
## iter  20 value 4693758.259355
## iter  20 value 4693758.248455
## final  value 4693758.248455
## converged
```

```
train_rmsr <- data.frame(cbind(1:20, train_rmsr))
ggplot(train_rmsr, aes(x = V1, y = train_rmsr)) + geom_point(size = 1, alpha = 1, na.rm = TRUE)
```

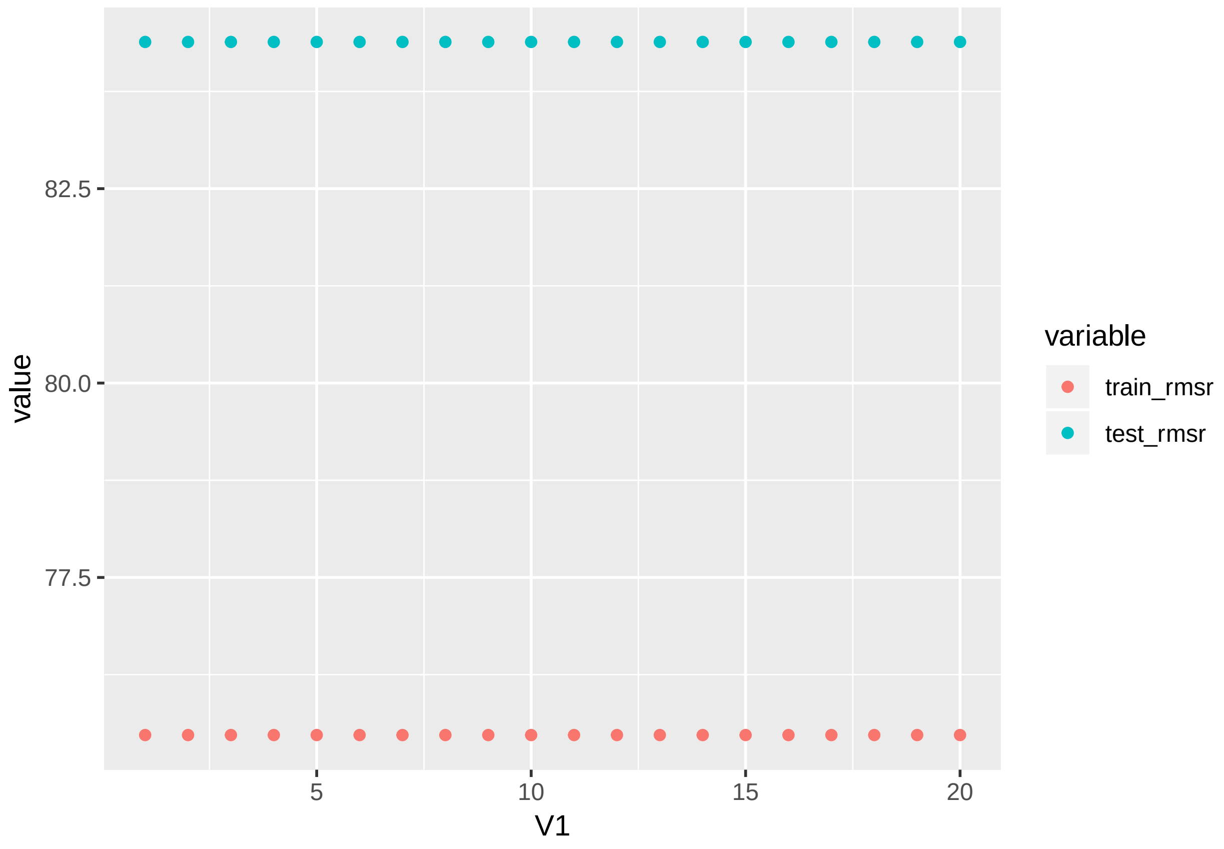


c) For the networks in b), compute also the root mean squared residuals on the test data and plot them in the same graph.

```
test_rmsr <- numeric()

for(i in 1:20) {
  nn_fit <- nnet(CCS ~ ., data= concrete_train, maxit = 2000, decay = .01,  size = i)
  test_rmsr[i] <- sqrt(mean((predict(nn_fit, newdata = concrete_test, type = "r") - concrete_test$CCS)^
}
```

```
## # weights:  11
## initial  value 4729302.365435
## iter  10 value 4694227.216250
## iter  20 value 4693775.670164
## final  value 4693760.472159
## converged
## # weights:  21
## initial  value 4710799.743444
## iter  10 value 4694012.589964
## iter  20 value 4693773.543211
## iter  30 value 4693759.153074
## final  value 4693759.055387
## converged
## # weights:  31
## initial  value 4719422.126321
## iter  10 value 4694359.610637
## iter  20 value 4693784.513689
## final  value 4693759.009337
## converged
## # weights:  41
## initial  value 4730838.963614
## iter  10 value 4693835.444067
## iter  20 value 4693758.855832
## final  value 4693758.746229
## converged
## # weights:  51
## initial  value 4740178.195552
## iter  10 value 4694735.085502
## iter  20 value 4693804.246611
## iter  30 value 4693760.838899
## final  value 4693759.306689
## converged
## # weights:  61
## initial  value 4730198.985032
## iter  10 value 4694725.574800
## iter  20 value 4693803.296897
## iter  30 value 4693759.157988
## final  value 4693758.715067
## converged
## # weights:  71
## initial  value 4723961.866460
## iter  10 value 4693764.683026
## iter  20 value 4693759.108100
## final  value 4693758.717296
## converged
## # weights:  81
## initial  value 4735191.344958
## iter  10 value 4693759.338280
## final  value 4693758.408167
## converged
## # weights:  91
## initial  value 4719882.126359
## iter  10 value 4693767.940595
## final  value 4693758.496051
## converged
## # weights:  101
## initial  value 4744872.587199
## iter  10 value 4694906.345843
## iter  20 value 4693831.173164
## iter  30 value 4693759.391376
## final  value 4693758.544459
## converged
## # weights:  111
## initial  value 4743757.813350
## iter  10 value 4693764.333856
## final  value 4693758.455412
## converged
## # weights:  121
## initial  value 4724116.759807
## iter  10 value 4693758.876623
## iter  10 value 4693758.856149
## iter  10 value 4693758.821910
## final  value 4693758.821910
## converged
## # weights:  131
## initial  value 4737240.418721
## iter  10 value 4693873.534684
## iter  20 value 4693762.516431
## final  value 4693758.454293
## converged
## # weights:  141
## initial  value 4720331.678715
## iter  10 value 4693769.375464
## final  value 4693759.262788
## converged
## # weights:  151
## initial  value 4714594.401830
## iter  10 value 4695468.759959
## iter  20 value 4693782.548144
## final  value 4693758.513508
## converged
## # weights:  161
## initial  value 4739409.487775
## iter  10 value 4693760.490039
## final  value 4693758.552958
## converged
## # weights:  171
## initial  value 4709730.224177
## iter  10 value 4693833.117051
## final  value 4693758.540616
## converged
## # weights:  181
## initial  value 4726750.545799
## iter  10 value 4693761.121054
## final  value 4693758.283411
## converged
## # weights:  191
## initial  value 4717060.392373
## iter  10 value 4693763.862299
## final  value 4693758.360945
## converged
## # weights:  201
## initial  value 4711377.296717
## iter  10 value 4693811.257787
## final  value 4693758.211858
## converged
```

```r
rmsr <- data.frame(cbind(train_rmsr, test_rmsr))

# melt the data to a long format
rmsr <- melt(data = rmsr, id.vars = "V1")

# plot
ggplot(data = rmsr, aes(x = V1, y = value, colour = variable)) + geom_point()
```



d) Is there evidence of overfitting? How can you tell?

When placing the RMSR for the train and test on the same plot, it is clear that there is an overfitting problem. The test set RMSR is consistently higher than the train RMSR, meaning that our model performs worse when applied to new data.

e) Do you think that the ANN is overfitting the data?

ANN appears to be overfitting the data. Even when there is just one node, the RMSR is much higher than that of the test set.