512: Homework 5

Brody Vogel

1. FeedFoward (5 points)

Below is the output from nnet after we fit a model. Let's assume we used a tanh() activation function throughout. Let $x_i, i = 1, 2, \ldots$ be the input variables and let $h_1, h_2, … $ 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
```

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

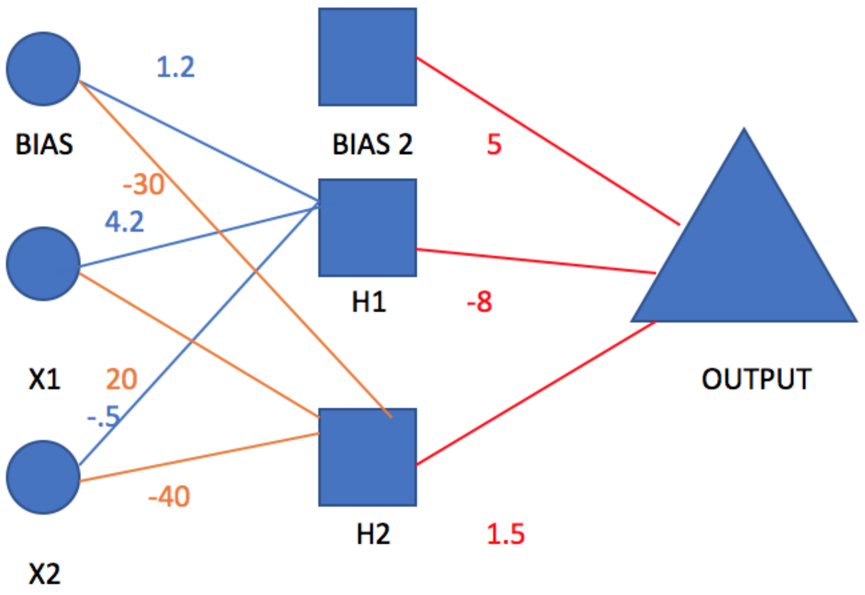

Figure 1: Neural Network Diagram

(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 $h_1 = f(x_1, x_2, \ldots)$ for a suitable explicit function f.*

$h_1 = f(1.2 + 4.2X_1 - .5X_2)$

$h_1 = tanh(1.2 + 4.2X_1 - .5X_2) <==> h_1 = \frac{e^{1.2+4.2X_1-.5X_2} - e^{-(1.2+4.2X_1-.5X_2)}}{e^{1.2+4.2X_1-.5X_2} + e^{-(1.2+4.2X_1-.5X_2)}}$

(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 $z = g(h_1, h_2, \dots)$ for a suitable explicit function g.*

$z = g(5 - 8h_1 + 1.5h_2)$

$z = tanh(5 - 8h_1 + 1.5h_2) <==> z = \frac{e^{5-8h_1+1.5h_2} - e^{-(5-8h_1+1.5h_2)}}{e^{5-8h_1+1.5h_2} + e^{-(5-8h_1+1.5h_2)}}$

(d) Provide an expression for the value at the output node as a function of the input values. *This should have the form $z = F(x_1, x_2, \dots)$ for a suitable explicit function F.*

$z = F(5 + f_1(1.2 + 4.2X_1 - .5X_2) + f_2(-30 + 20X_1 - 40X_2))$

$z = tanh(5 - 8tanh(1.2 + 4.2X_1 - .5X_2) + 1.5tanh(-30 + 20X_1 - 40X_2)) <==> z = tanh(5 - 8(\frac{e^{1.2+4.2X_1-.5X_2} - e^{-(1.2+4.2X_1-.5X_2)}}{e^{1.2+4.2X_1-.5X_2} + e^{-(1.2+4.2X_1-.5X_2)}}) + 1.5(\frac{e^{-30+20X_1-40X_2} - e^{-(-30+20X_1-40X_2)}}{e^{-30+20X_1-40X_2} + e^{-(-30+20X_1-40X_2}})) <==> z = \frac{e^{5-8(\frac{e^{1.2+4.2X_1-.5X_2} - e^{-(1.2+4.2X_1-.5X_2)}}{e^{1.2+4.2X_1-.5X_2} + e^{-(1.2+4.2X_1-.5X_2)}})+1.5}}{e^{5-8(\frac{e^{1.2+4.2X_1-.5X_2} - e^{-(1.2+4.2X_1-.5X_2)}}{e^{1.2+4.2X_1-.5X_2} + e^{-(1.2+4.2X_1-.5X_2)}})+1.5}}$

2. Multiple Solutions (5)

Below is the nnet putput for a feed-forward ANN that is similar to the network used in the videos.

```
a 2-2-1 network with 9 weights
options were - linear output units
 b->h1 i1->h1 i2->h1
  0      5      0
 b->h2 i1->h2 i2->h2
  0      0      7
  b->o  h1->o  h2->o
  -5      4      6
```

There are several other sets of weights that lead to networks with the exact same output behavior. Find **all** these sets of weights, with explanation.

- In the input to the two nodes in the hidden layer, as long as the value $5X_1$ is passed to h_1 and $7X_2$ to h_2, the output will be the same. Since we have a free bias (β_0) from the input layer at our disposal, we can use it to change the weights. Thus, any of the following will work:
- $b- > h1 = -10X_1, b- > h2 = -15X_2$, $i1- > h1 = 15$, and $i2- > h2 = 22$
- $b- > h1 = 100X_1, b- > h2 = 150X_2$, $i1- > h1 = -95$, and $i2- > h2 = -143$
- $b- > h1 = 5X_1, b- > h2 = 7X_2$, $i1- > h1 = 0$, and $i2- > h2 = 0$
- so on and so forth
- A similar thing can be done in the input from the hidden layer to the outer layer. As long as there is a -5 in the bias term, $4h_1$, and $6h_2$, all being inputted to the output function, we'll get the same results. Once more, any of the following will work:
- $b- > O = -5 - 10X_1 - 15X_2$, $h1- > O = 14$, $h2- > O = 21$
- $b- > O = -5 + 100X_1 + 150X_2$, $h1- > O = -96$, $h2- > O = -144$

- $b- > O = -5 + 4X_1 + 6X_2$, $h1- > O = 0$, $h2- > O = 0$
- so on and so forth
- Finally, any combination of the last two bullets will also produce the same output. So, for example, the following will work:
- $b- > h1 = -10X_1, b- > h2 = -15X_2$, $i1- > h1 = 15$, and $i2- > h2 = 22$, $b- > O = -5-10X_1-15X_2$, h1 -> O = 14, h2 -> O = 21$,

Thus, any combination of weights for the ANN in which the value $5X_1$ is passed to h_1, $7X_2$ to h_2, and $-5 + 4h_1 + 6h_2$ to the output layer will produce results identical to that of the given network.

3. ANNs and Multiple Linear Regression (7 points)

This problem uses the Advertising data. All data should be used for training. *When using nnet() in this problem, leave **decay** and **maxit** at their default values.*

```
library(nnet)
set.seed(1234)
Advertising <- read.csv("/Users/brodyvogel/Desktop/Advertising.csv")
```

A linear regression model Sales ~ TV + Radio + Newspaper can be fitted using either R's lm() function or with a neural net without hidden layer and with linear output. Carry this out and demonstrate that the models are the same (same coefficients, same predictions).

```
lm.fit <- lm(Sales~TV+Radio+Newspaper, data = Advertising)
ann.1 <- nnet(Sales~TV+Radio+Newspaper, data = Advertising, size = 0, linout = TRUE, skip = TRUE)
```

```
## # weights:  4
## initial  value 102721.171716
## final  value 556.825263
## converged
```

```
preds_match <- round(lm.fit$fitted.values, 1) == round(ann.1$fitted.values, 1)

FALSE %in% preds_match
```

```
## [1] FALSE
```

```
#summary(lm.fit)
#summary(ann.1)
```

The coefficients in both models are ≈: $\hat{\beta_0} = 2.94, \hat{\beta_{TV}} = .05, \hat{\beta_{Radio}} = .19, \hat{\beta_{Newspaper}} = 0$.

Since the models have the same form and coefficients, they will produce the same predictions, too. This is confirmed above, since there are no differing predictions produced by the models' fitted values.

(a) Demonstrate that the lm() and the nnet() approach essentially give the same model. Also show that the sum of squared errors (SSE) is the same for both models.

As can be seen, both approaches give approximately the same model: $\hat{Y} = 2.94 + .05 \times TV + .19 \times Radio + 0 \times Newspaper$. Since the models and data inputted to the models are the same, we know the SSEs will also be the same, which is confirmed below: $SSE_{lm} = SSE_{ann} = 556.8253$.

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper, data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8277 -0.8908  0.2418  1.1893  2.8292
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.938889   0.311908   9.422   <2e-16 ***
## TV           0.045765   0.001395  32.809   <2e-16 ***
## Radio        0.188530   0.008611  21.893   <2e-16 ***
## Newspaper   -0.001037   0.005871  -0.177     0.86
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.686 on 196 degrees of freedom
## Multiple R-squared:  0.8972, Adjusted R-squared:  0.8956
## F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

```
summary(ann.1)
```

```
## a 3-0-1 network with 4 weights
## options were - skip-layer connections  linear output units
##  b->o i1->o i2->o i3->o
##  2.94  0.05  0.19  0.00
```

```
SSE.lm <- sum(lm.fit$residuals**2)
SSE.ann.1 <- sum(ann.1$residuals**2)

SSE.lm
```

```
## [1] 556.8253
```

```
SSE.ann.1
```

```
## [1] 556.8253
```

(b) Use the `scale()` method to create a new dataframe **Advert2** from the Advertising data. Use ‘head()’ to inspect this dataframe and describe what has changed. Then replace the changed **Sales** column with the original one, since we do not want to change the responses.

When rescaleing the feature space, one calculates the mean and variance of each feature and converts each measurement into a feature-specific z score. This is generally useful if the different features are measured on entirely different units or scales.

```
Advert2 <- data.frame(scale(Advertising))

#head(Advertising)
head(Advert2)
```

```
##           X          TV     Radio Newspaper      Sales
## 1 -1.719098  0.96742460  0.9790656 1.7744925  1.5481681
## 2 -1.701821 -1.19437904  1.0800974 0.6679027 -0.6943038
## 3 -1.684543 -1.51235985  1.5246374 1.7790842 -0.9051345
## 4 -1.667266  0.05191939  1.2148065 1.2831850  0.8581768
## 5 -1.649989  0.39319551 -0.8395070 1.2785934 -0.2151431
## 6 -1.632711 -1.61136487  1.7267010 2.0408088 -1.3076295
```

```r
Advert2$Sales <- Advertising$Sales
```

Each feature of the dataframe, now, is on the same scale; that is, the variables have been 'standardized'. So, for instance, the mean of TV will be roughly the same as that of Radio and Newspaper. In actuality, the function has changed the value in each cell of the dataframe to reflect the number of standard deviations that value is from its respective column's mean.

(c) Fit a linear model with one of the methods of (a) to the scaled data. Explain why this linear model will have the exact same responses as the linear model from (a).

```r
lm.fit2 <- lm(Sales~TV+Radio+Newspaper, data = Advert2)

preds_match1 <- round(lm.fit2$fitted.values, 1) == round(lm.fit$fitted.values, 1)

FALSE %in% preds_match1
```

```
## [1] FALSE
```

From above, we can see that the responses are the same in this new linear model as in the first. This is the case, because, in both models, the coefficients are calculated to minimize the same loss function; i.e., the difference between the true Sales value and what the models predict. Therefore, in lm.fit2, because the values of the predictors are scaled, the values of the coefficients mirror that scaling. So the scaled change from lm.fit to lm.fit2 in the predictor values is echoed/cancelled out in the scaled change in the models' coefficients. Thus, there is a change, and a kind of 'change back', interplay between the predictor and coefficient values in the models, and so their respective fitted values are the same.

4. MNIST Revisited (8 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
mnist <- load('/Users/brodyvogel/Downloads/mnist_all.RData')

z1 <- which(train$y %in% c(4,7))

my.train.x <- train$x[z1, ]

my.train.y <- train$y[z1]
```

(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.

I picked variables 266 and 401. They had the 7th and 8th largest variances in the training set, and the correlation between them was only -.325. My model did fairly well; I got an AUC of .8345. And, unlike in the previous assignment, there was a fair bit of variability in the training data. My model tended to overestimate the the number of 7s and underestimate the 4s, it looks like, which somewhat explains my only getting an AUC of .8345.

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
sub <- order(-apply(my.train.x, 2, var))[1:10]

#sub
#cor(my.train.x[, sub])

my.train.y <- as.numeric(my.train.y > 5)

log.fit <- glm(my.train.y ~ my.train.x[ , 266] + my.train.x[ , 401], family = binomial)

probs <- predict(log.fit, data.frame(my.train.x), type = 'response')

pred <- rep(0, length(probs))

pred[probs > .5] = 1

r <- roc(my.train.y, pred)

#plot(r)

auc(my.train.y, pred)
```

```
## Area under the curve: 0.8345
```

```r
#table(pred, my.train.y)
```

(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.

I got almost identical results. The AUC, this time, was .8356, as compared to .8345 in the logistic model. This makes a fair bit of sense, because, if we think of the hidden layer in this ANN as a logistic transformation of the input, then this model is essentially identical to the logistic model we fit in the previous question.

```r
my.ann.1 <- nnet(my.train.y ~ my.train.x[ , 266] + my.train.x[ , 401], size = 1)
```

```
## # weights:  5
## initial  value 2887.472909
## iter  10 value 1648.660680
## iter  20 value 1549.356380
## iter  30 value 1478.727554
## iter  40 value 1459.141250
## final  value 1459.003088
## converged
```

```
probs <- predict(my.ann.1, type = 'raw')

pred <- rep(0, length(probs))

pred[probs > .5] = 1

r <- roc(my.train.y, pred)

#plot(r)

auc(my.train.y, pred)
```

```
## Area under the curve: 0.8338
```

```
table(pred, my.train.y)
```

```
##     my.train.y
## pred    0    1
##    0 4503  647
##    1 1339 5618
```

(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? Is there evidence of overfitting?

I got AUCs of .8328, .8392, and .8436 for 5, 10, and 12 hidden layers, respectively. So, then, adding more hidden layers didn't much improve my ANN. The models' predictions didn't much change with the addition of more hidden layers, either. Just from the AUC and predictions, there's no way to be *certain* that the models have overfitted; but the fact that there was little change in the AUC and predictions with a 3 fold increase in the number of hidden layers makes me think the models have extracted every nuance from the training data, and so - yes - have overfitted to said data.

```
for (x in c(5, 10, 12)) {
  my.ann.2 <- nnet(my.train.y ~ my.train.x[ , 266] + my.train.x[ , 401], size = x, maxit = 1000)

  probs <- predict(my.ann.2, type = 'raw')

  pred <- rep(0, length(probs))

  pred[probs > .5] = 1

  r <- roc(my.train.y, pred)

  #plot(r)

  print(auc(my.train.y, pred))

  print(table(pred, my.train.y))
}
```

```
## # weights:  21
## initial  value 2986.323695
## iter  10 value 1519.538421
## iter  20 value 1503.572759
## iter  30 value 1503.479550
## iter  40 value 1503.273578
## iter  50 value 1503.236055
## iter  60 value 1503.231367
## iter  70 value 1503.230135
## iter  80 value 1503.229198
## final  value 1503.228294
## converged
## Area under the curve: 0.8328
##     my.train.y
## pred    0    1
##    0 4805  983
##    1 1037 5282
## # weights:  41
## initial  value 2890.798661
## iter  10 value 1678.057991
## iter  20 value 1636.302287
## iter  30 value 1623.031156
## iter  40 value 1468.137683
## iter  50 value 1443.940571
## iter  60 value 1422.567403
## iter  70 value 1404.761056
## iter  80 value 1358.826006
## iter  90 value 1352.265165
## iter 100 value 1349.598789
## iter 110 value 1348.740080
## iter 120 value 1347.415612
## iter 130 value 1346.982853
## iter 140 value 1346.484294
## iter 150 value 1346.012765
## iter 160 value 1345.846473
## iter 170 value 1345.646476
## iter 180 value 1345.535109
## final  value 1345.486659
## converged
## Area under the curve: 0.8392
##     my.train.y
## pred    0    1
##    0 4695  785
##    1 1147 5480
## # weights:  49
## initial  value 4825.746843
## iter  10 value 1625.703586
## iter  20 value 1421.298227
## iter  30 value 1414.539870
## iter  40 value 1407.630609
## iter  50 value 1390.870939
## iter  60 value 1388.598978
## iter  70 value 1386.938827
## iter  80 value 1384.614809
## iter  90 value 1378.507966
## iter 100 value 1366.910473
## iter 110 value 1351.579145
## iter 120 value 1348.634171
## iter 130 value 1347.480416
## iter 140 value 1345.695827
## iter 150 value 1345.049761
## iter 160 value 1343.837898
## iter 170 value 1343.474839
## iter 180 value 1343.283876
## final  value 1343.208675
## converged
## Area under the curve: 0.8413
##     my.train.y
## pred    0    1
##    0 4682  745
##    1 1160 5520
```