Homework IV Data Mining II

Allen Rahrooh

Problem 11.1

Establish the exact correspondance between the projection pursuit regression model (11.1) and the neural network (11.5). In particular, show that the single-layer regression network is equivalent to a PPR model with $g_m(w_m^T x) = \beta_m \sigma(\alpha_{0m} + s_m(w_m^T x))$ where w_m is the mth unit vector. Establish a similiar equivalence for a classification network.

We start with a neural net with $k = 1$. Using Eq. 11.5 we get:

$$f(X) = g(T) = g(\beta_0 + \beta Z) = g(\beta_0 + \sum_m \beta_m z_m)$$

We take β_0 as a constant term which is a column of ones in X and with the assumption that $g(*)$ is an indentity function:

$$f(X) = \sum_m \beta_m \sigma(\alpha_{0m} + \alpha_m^T X)$$

Doing a comparison to Eq. 11.1:

$$g_m(w_m^T X) = \beta_m \sigma(\alpha_{0m} + ||\alpha_m|| \frac{\alpha_m^T}{||\alpha_m||} X)$$

where $\frac{\alpha_m^T}{||\alpha_m||}$ is a unit vector. $s_m = ||\alpha_m||$.

Problem 11.2

Consider a neural network for a quantitative outcome as in (11.5), using squared-error loss and identity output function $g_k(t) = t$. Suppose that the weights α_m from the input to hidden layer are nearly zero. Show that the resulting model is nearly linear in the inputs.

We start with the Taylor expansion of the activation function $\sigma(v)$.

$$\sigma(\alpha_{0m} + \alpha_m^T X) = \frac{1}{1 + exp(-\alpha_{0m} - \alpha_m^T X)}$$

$$\frac{e^{\alpha_{0m}}}{e^{\alpha_{0m}} + e^{-\alpha_m^T X}}$$

$$(\frac{e^{\alpha_{0m}}}{1 + e^{\alpha_{0m}}})(\frac{1}{\frac{e^{-\alpha_m^T X} - 1}{1 + e^{\alpha_{0m}}} + 1})$$

If $X \to 0$:

$$e^{-\alpha_m^T X} - 1 \to -\alpha_m^T X$$

$$\sigma(\alpha_{0m} + \alpha_m^T X) \to (\frac{e^{\alpha_{0m}}}{1+e^{\alpha_{0m}}})(1 + \frac{\alpha_m^T X}{1+e^{\alpha_{0m}}})$$

Problem 11.5

(a) Write a program to fit a single hidden layer neural network (ten hidden units) via back-propagation and weight decay.

(b) Apply it to 100 observations from the model

$$Y = \sigma(\alpha_1^T X) + (\alpha_2^T X)^2 + 0.30Z$$

where *sigma* is the sigmoid function, Z is the standard normal, $X^T = (X_1, X_2)$, each X_j being independent standard normal, and $\alpha_1 = (3, 3), \alpha_2 = (3, -3)$. Generate a test sample of size 1000, and plot the training and test error curves as a function of the number of training epochs, for different values of the weight decay parameter. Discuss the overfitting behavior in each case.

Defining model and creating training and testing sets.

```
library(MASS)
library(caret)
library(neuralnet)
library(nnet)
library(readr)

#11.5
Y_Observations <- function(n){
  X1 = rnorm(n, 0, 1)
  X2 = rnorm(n, 0, 1)
  XT = cbind(X1, X2)
  X = t(XT)
  alpha1 = c(3,3)
  alpha2 = c(3,-3)
  Y = c(1/(1+exp(-(alpha1%*%X)))+ (alpha2%*%X)**2 + 0.3*rnorm(n, 0, 1))
  Observations = cbind(X1, X2, Y)}

#training sample of 100 samples
Y_train <- as.data.frame(Y_Observations(100))

#test sample of 1000 samples
Y_test <- as.data.frame(Y_Observations(1000))
```

```
# fit neural network
nn <- nnet(Y~., data=Y_train, size =10, decay=0.1)
```

```
## # weights:  41
## initial  value 85955.024619
## iter  10 value 84099.533749
## iter  20 value 84099.133053
## final  value 84099.131092
## converged
```

```r
nn_predictions <- predict(object=nn, newdata=Y_test)
test_MSE <- sum((nn_predictions - Y_test)^2)/nrow(Y_test)

#Plotting
nn1 <- nnet(Y~., data=Y_train, size =10, decay=1)
```

```
## # weights:  41
## initial  value 85103.373894
## iter  10 value 84113.502665
## final  value 84112.133043
## converged
```

Decay of 10

test_MSE <- c()

train_MSE <- c()

for (i in 1:50){

nn <- nnet(Y~., data=Y_train, size =10, decay=10, maxit=i)

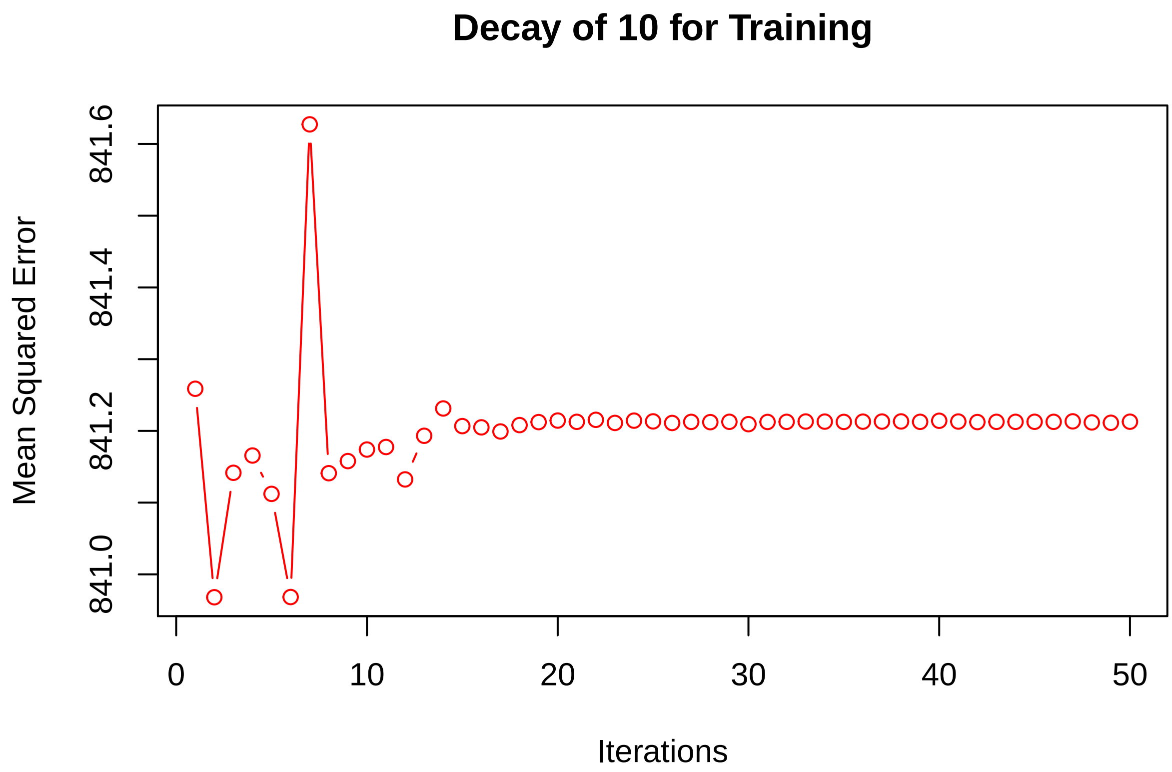
nn_predictions <- predict(object=nn, newdata=Y_test)

test_MSE <- append(test_MSE, sum((Y_test - nn_predictions)**2)/nrow(Y_test))

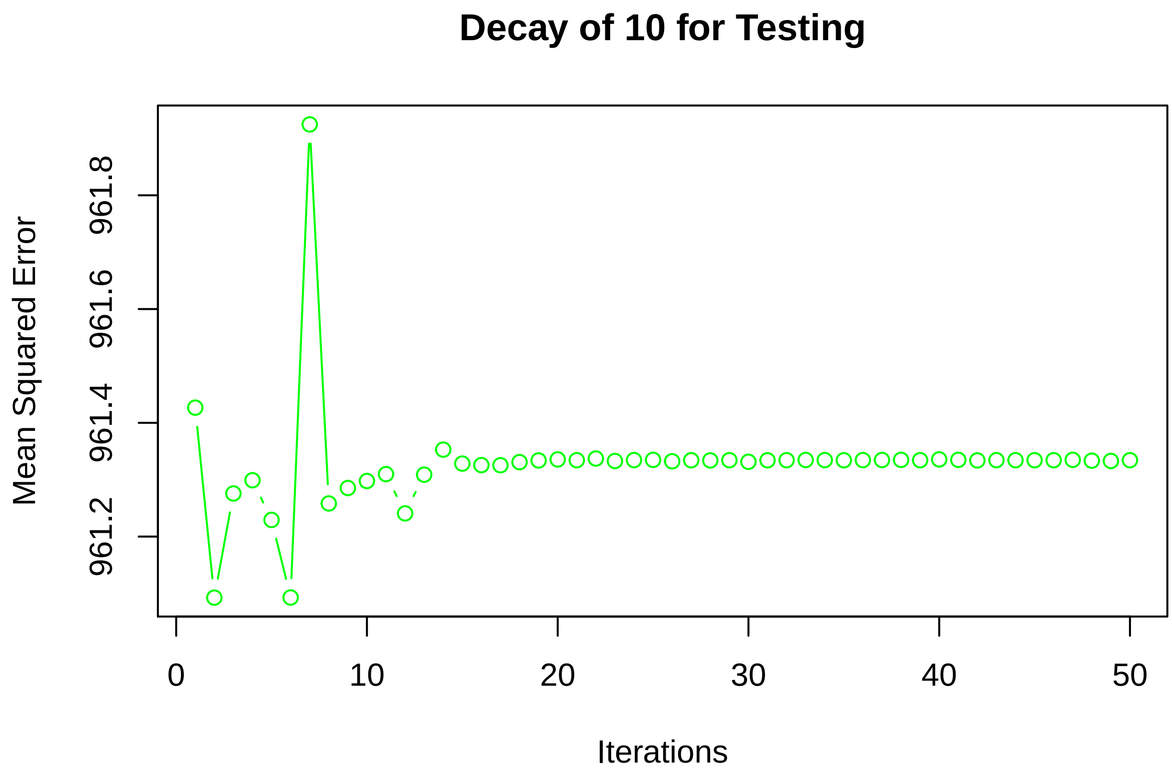
train_MSE <- append(train_MSE, mean(nn$residuals**2))

}

```r
#Plotting

plot(x= 1:50, y= train_MSE[1:50], type="b", xlab="Iterations", ylab= "Mean Squared Error",
     col="red", main = "Decay of 10 for Training")
```




```r
plot(x= 1:50, type="b", y= test_MSE[1:50], xlab="Iterations", ylab= "Mean Squared Error",
     col="green", main = "Decay of 10 for Testing")
```



Decay of 1

test_MSE <- c()

train_MSE <- c()

for (i in 1:50){

nn <- nnet(Y~., data=Y_train, size =10, decay=1, maxit=i)

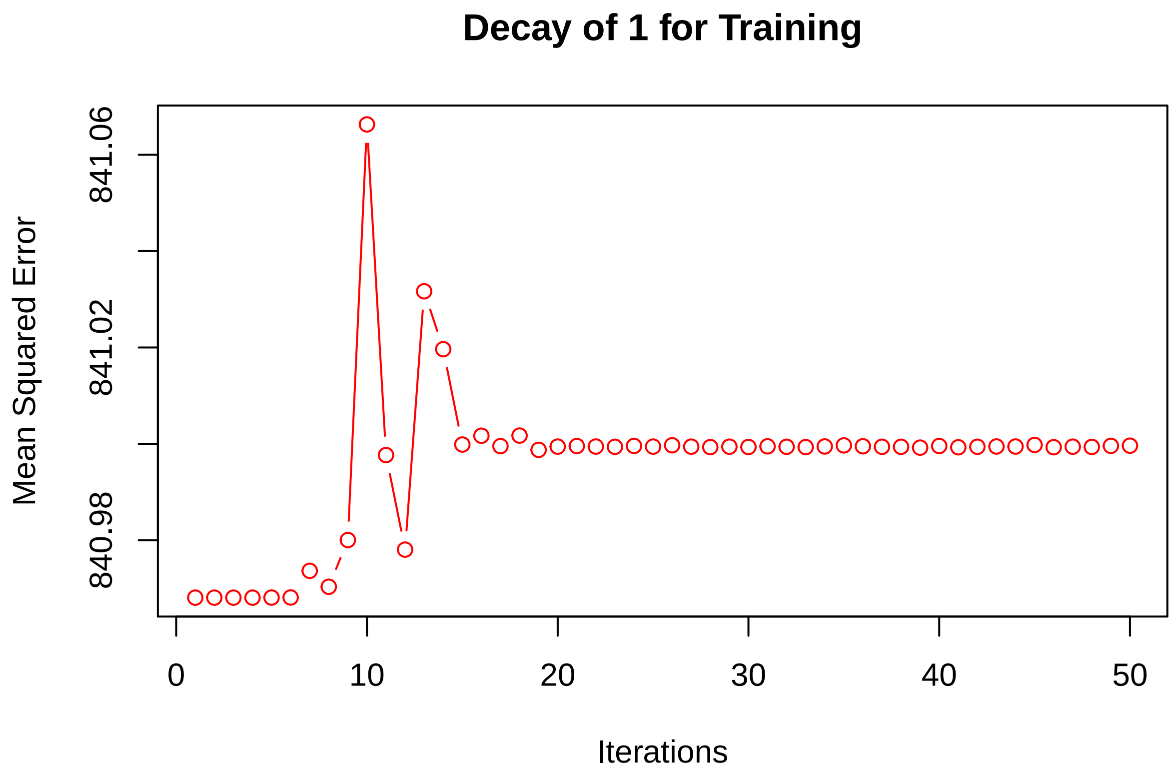
nn_predictions <- predict(object=nn, newdata=Y_test)

test_MSE <- append(test_MSE, sum((Y_test - nn_predictions)**2)/nrow(Y_test))

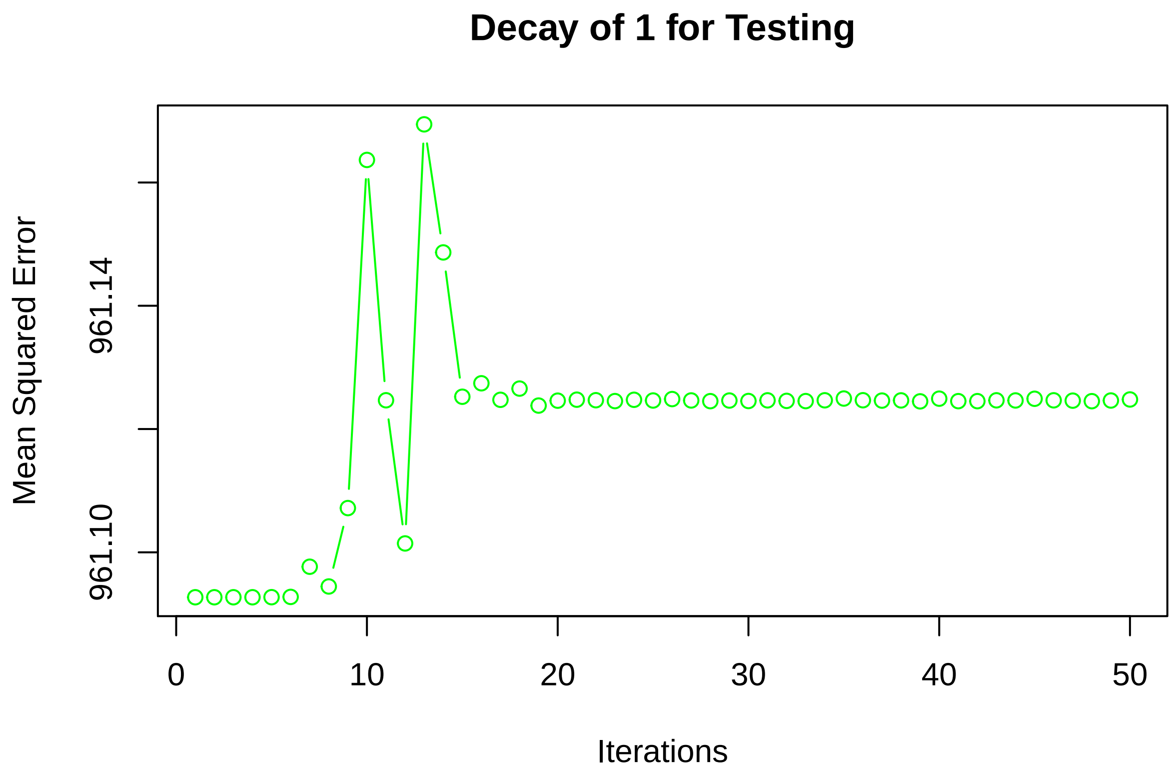
train_MSE <- append(train_MSE, mean(nn$residuals**2))

}

```
#Plotting

plot(x= 1:50, y= train_MSE[1:50], type="b", xlab="Iterations", ylab= "Mean Squared Error",
     col="red", main = "Decay of 1 for Training")
```




```
plot(x= 1:50, type="b", y= test_MSE[1:50], xlab="Iterations", ylab= "Mean Squared Error",
     col="green", main = "Decay of 1 for Testing")
```



Decay of 0.05

test_MSE <- c()

train_MSE <- c()

for (i in 1:50){

nn <- nnet(Y~., data=Y_train, size =10, decay=0.05, maxit=i)

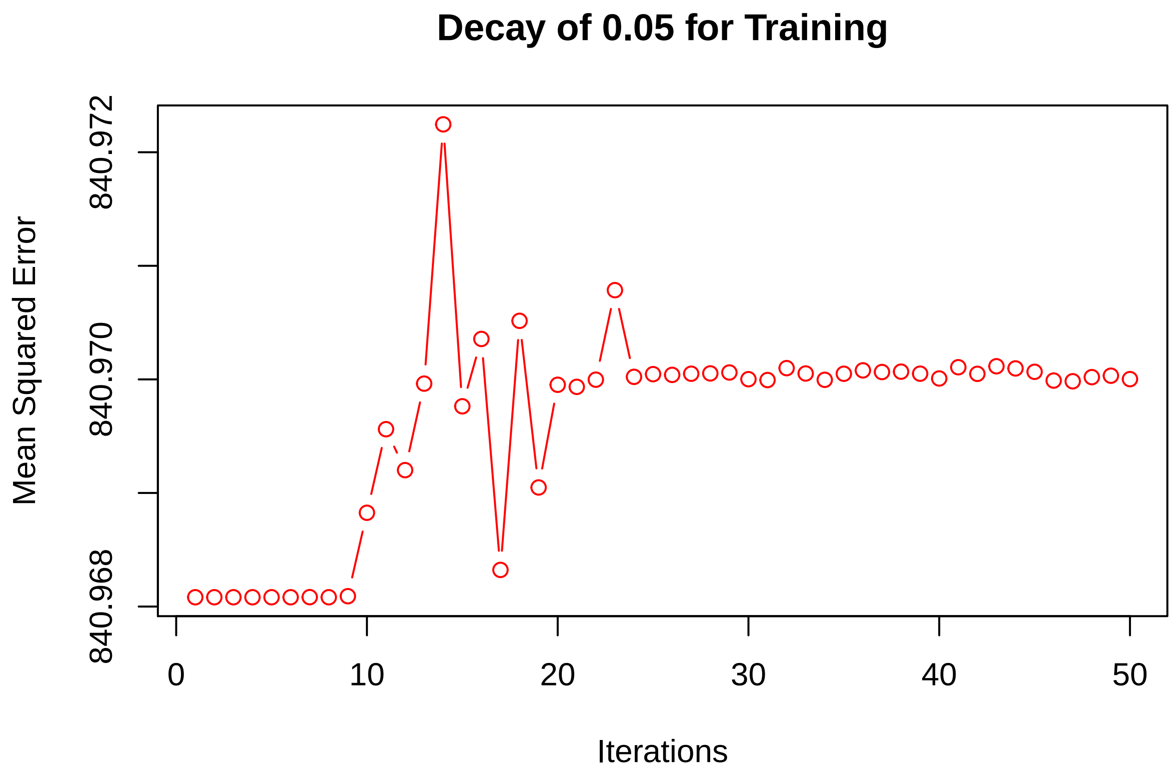
nn_predictions <- predict(object=nn, newdata=Y_test)

test_MSE <- append(test_MSE, sum((Y_test - nn_predictions)**2)/nrow(Y_test))

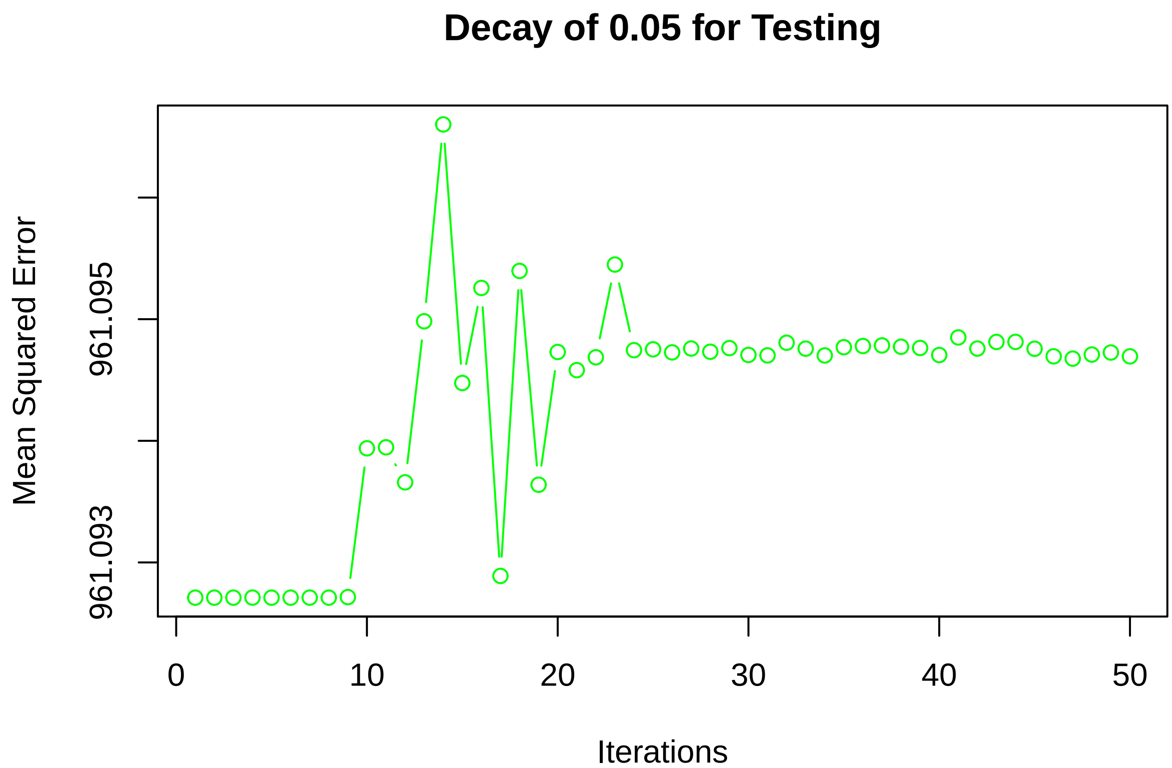
train_MSE <- append(train_MSE, mean(nn$residuals**2))

}

```r
#Plotting

plot(x= 1:50, y= train_MSE[1:50], type="b", xlab="Iterations", ylab= "Mean Squared Error",
     col="red", main = "Decay of 0.05 for Training")
```




```
plot(x= 1:50, type="b", y= test_MSE[1:50], xlab="Iterations", ylab= "Mean Squared Error",
     col="green", main = "Decay of 0.05 for Testing")
```



Decay of 0.01

test_MSE <- c()

train_MSE <- c()

for (i in 1:50){

nn <- nnet(Y~., data=Y_train, size =10, decay=0.01, maxit=i)

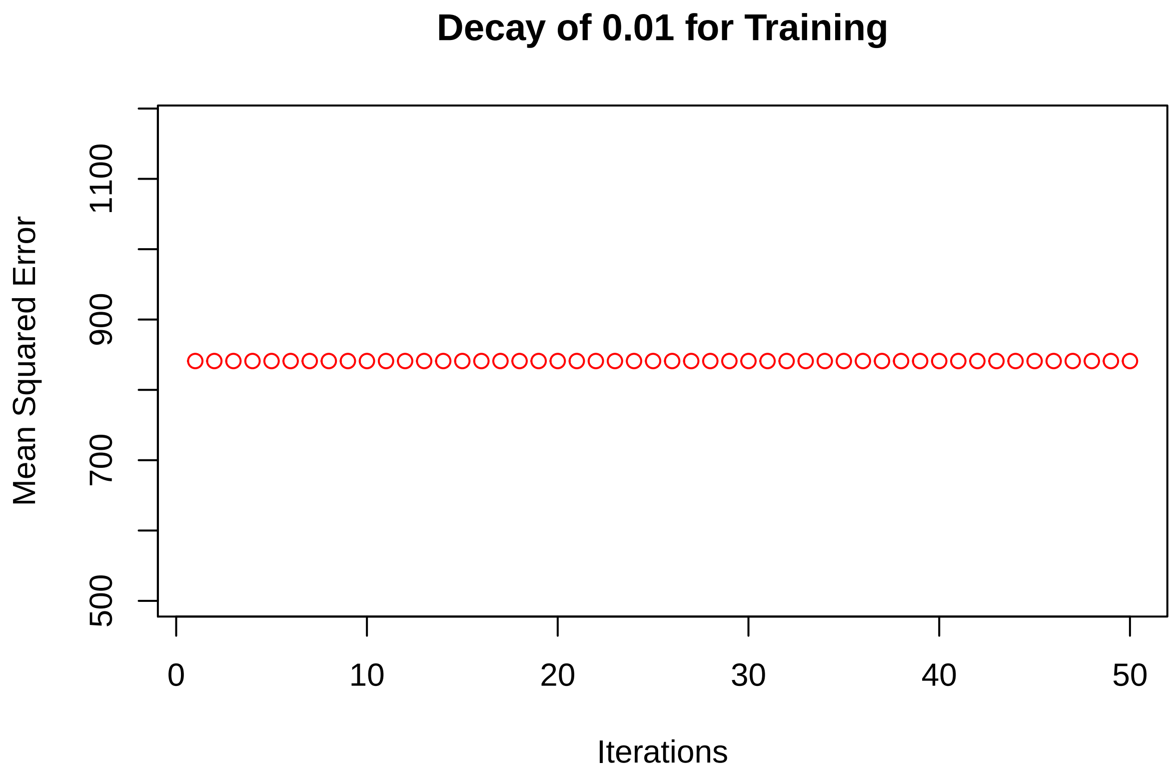
nn_predictions <- predict(object=nn, newdata=Y_test)

test_MSE <- append(test_MSE, sum((Y_test - nn_predictions)**2)/nrow(Y_test))

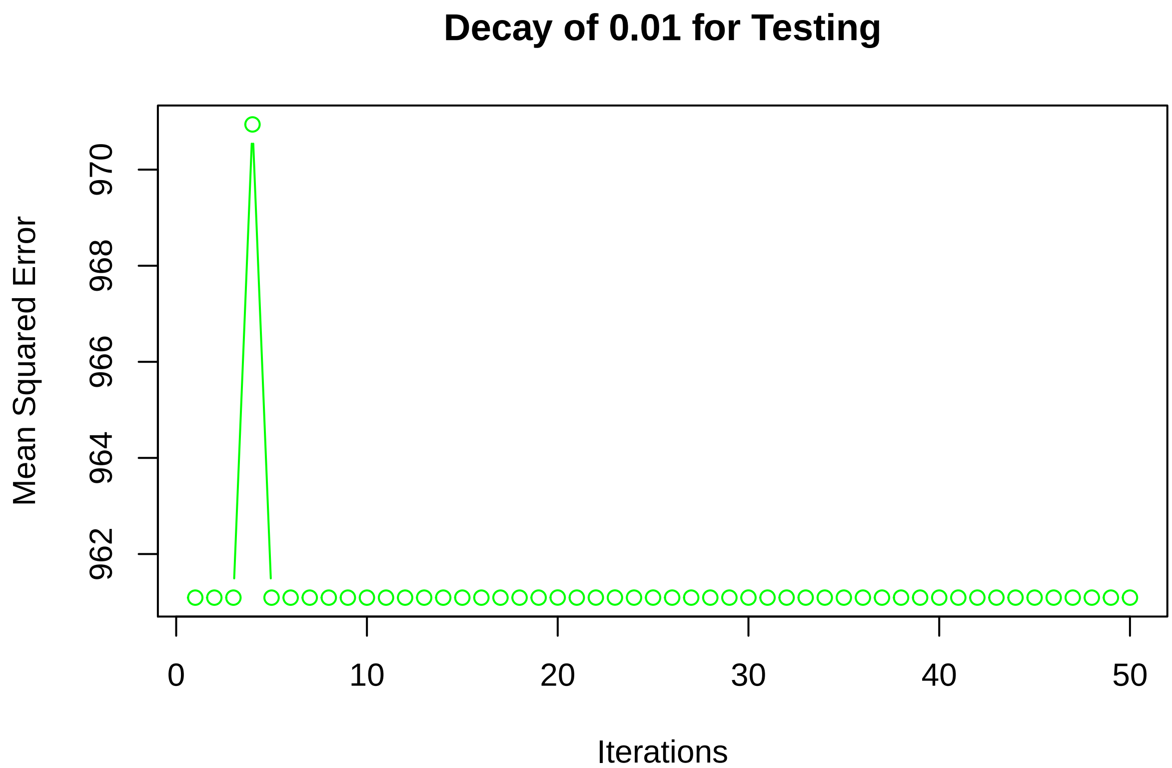
train_MSE <- append(train_MSE, mean(nn$residuals**2))

}

```
#Plotting

plot(x= 1:50, y= train_MSE[1:50], type="b", xlab="Iterations", ylab= "Mean Squared Error",
     col="red", main = "Decay of 0.01 for Training")
```




```
plot(x= 1:50, type="b", y= test_MSE[1:50], xlab="Iterations", ylab= "Mean Squared Error",
     col="green", main = "Decay of 0.01 for Testing")
```



Decay of 0.0001

test_MSE <- c()

train_MSE <- c()

for (i in 1:50){

nn <- nnet(Y~., data=Y_train, size =10, decay=0.0001, maxit=i)

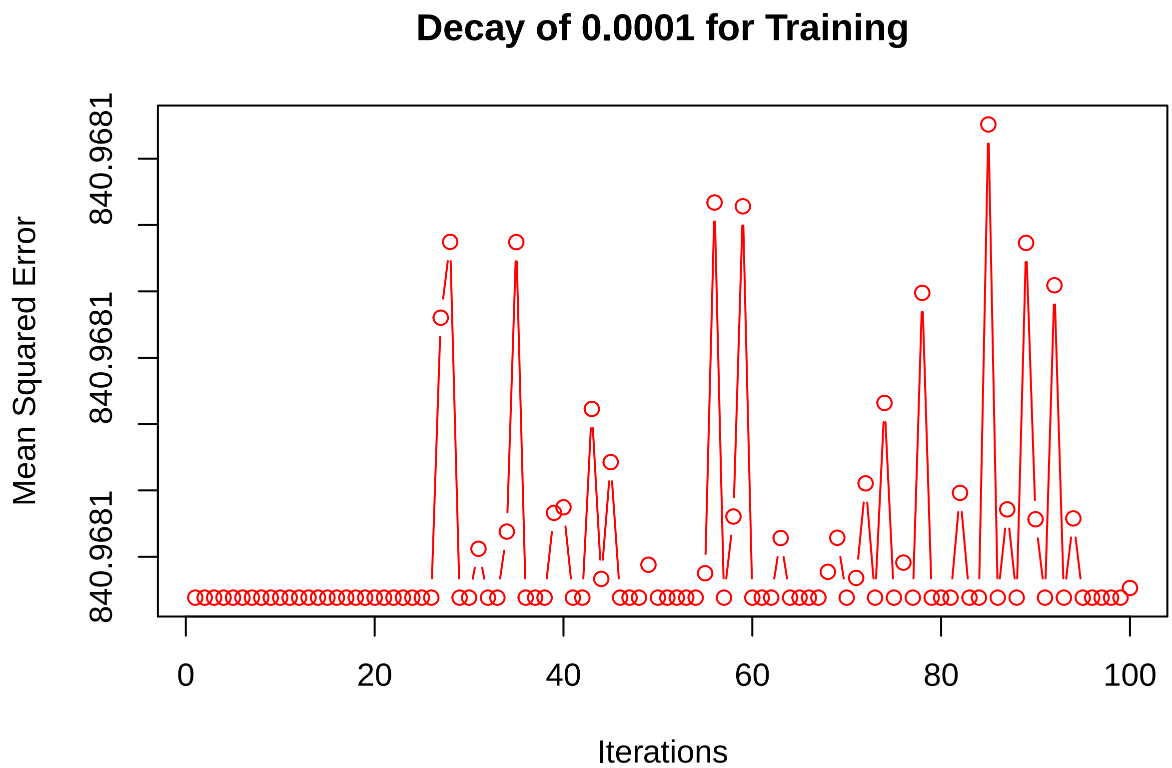
nn_predictions <- predict(object=nn, newdata=Y_test)

test_MSE <- append(test_MSE, sum((Y_test - nn_predictions)**2)/nrow(Y_test))

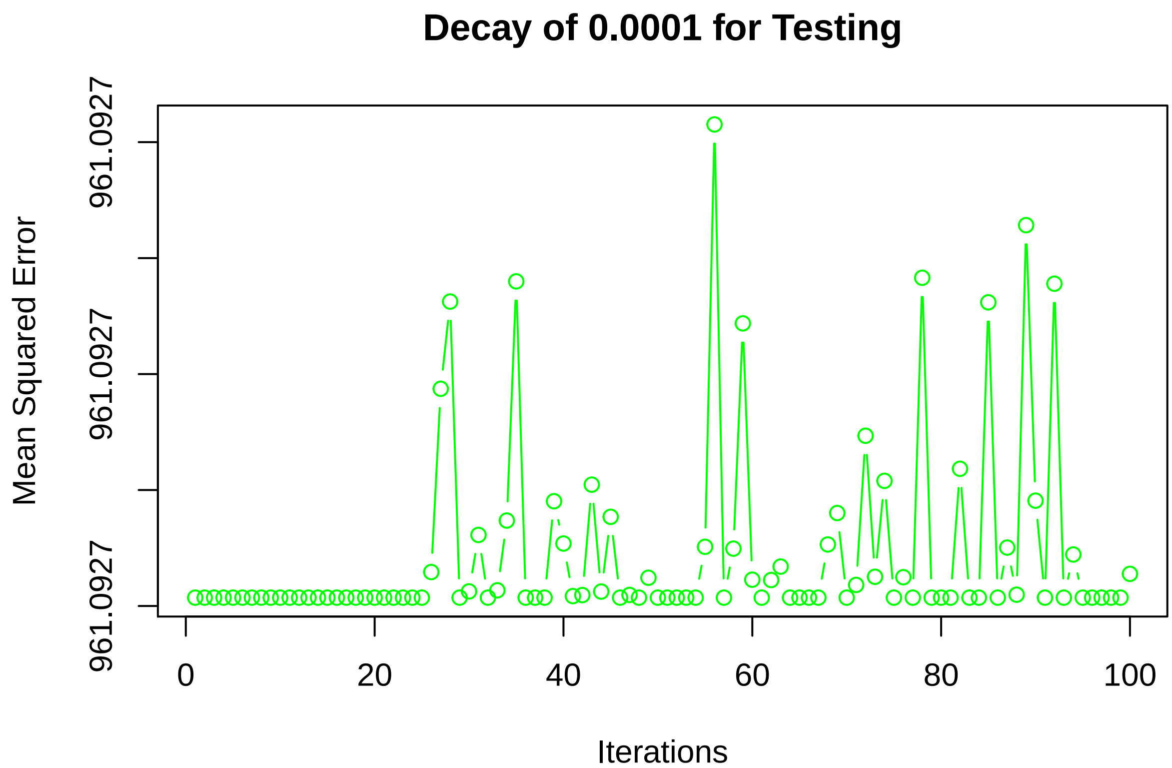
train_MSE <- append(train_MSE, mean(nn$residuals**2))

}

```
#Plotting

plot(x= 1:100, y= train_MSE[1:100], type="b", xlab="Iterations", ylab= "Mean Squared Error",
     col="red", main = "Decay of 0.0001 for Training")
```




```
plot(x= 1:100, type="b", y= test_MSE[1:100], xlab="Iterations", ylab= "Mean Squared Error",
     col="green",main = "Decay of 0.0001 for Testing")
```



Comparing all the decay plots it seems like with a decay of 0.01 the mean sqaured error stays constant unlike the other decay plots at values of 10, 1, 0.0001, and 0.05.

(c) Vary the number of hidden units in the network, from 1 up to 10, and determine the minimum number needed to perform well for this task.

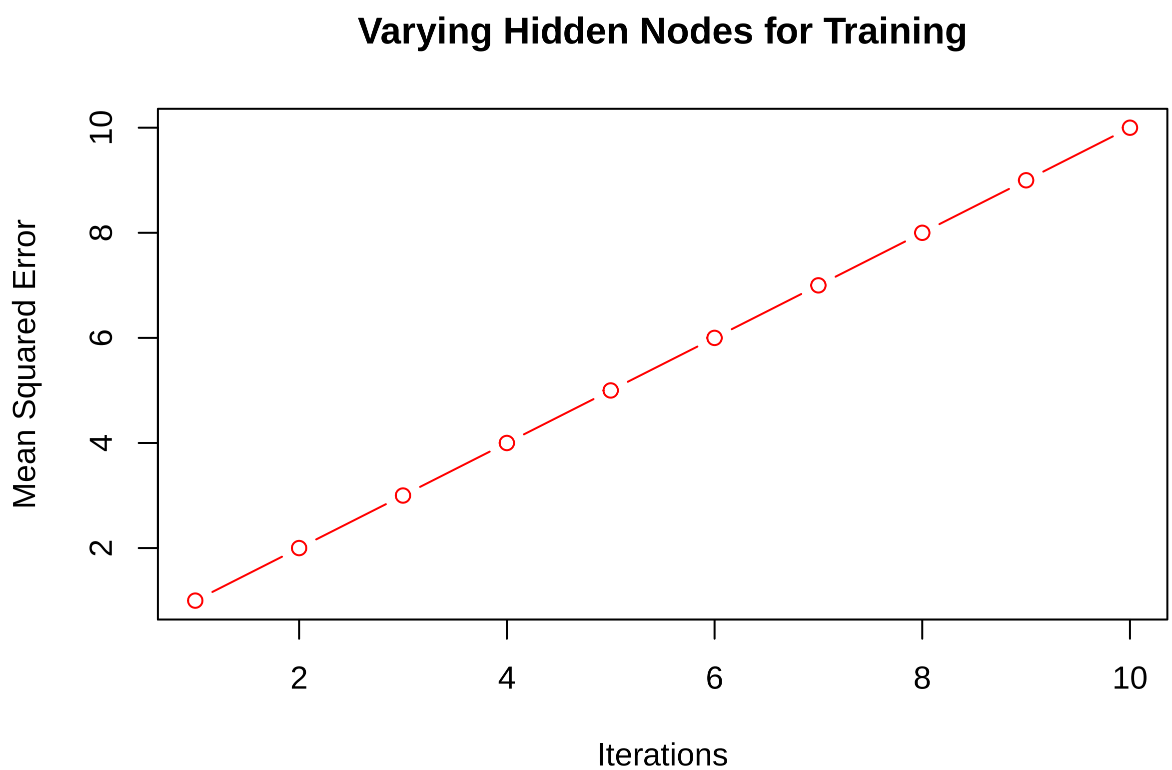
```
#Varying the hidden nodes
test_MSE <- c()
train_MSE <- c()
for (i in 1:10){
  nn <- nnet(Y~., data=Y_train, size =i, decay=0.01)
  nn_predictions <- predict(object=nn, newdata=Y_test)
  Test_MSE <- append(test_MSE, sum((Y_test - nn_predictions)**2)/nrow(Y_test))
  Train_MSE <- append(train_MSE, mean(nn1$residuals**2))
}
```

```
## # weights:  5
## initial  value 86173.309082
## iter  10 value 84111.817469
## iter  20 value 84097.706555
## iter  30 value 84097.559895
## final  value 84097.557651
## converged
## # weights:  9
## initial  value 85840.349955
## iter  10 value 84119.327426
```

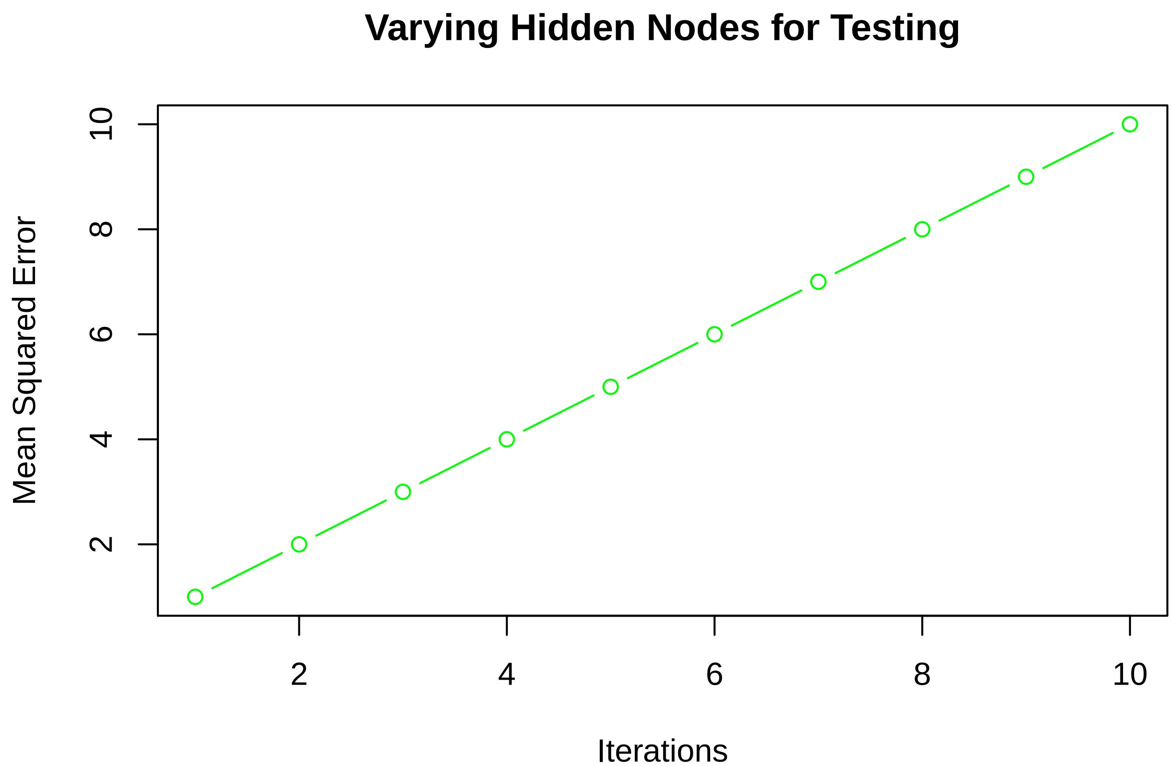
```
## iter  20 value 84097.889601
## iter  30 value 84097.450831
## final  value 84097.413230
## converged
## # weights:  13
## initial  value 86173.254226
## iter  10 value 84121.891808
## iter  20 value 84097.856915
## iter  30 value 84097.352289
## final  value 84097.336349
## converged
## # weights:  17
## initial  value 85466.353922
## iter  10 value 84126.014649
## iter  20 value 84097.850694
## iter  30 value 84097.314611
## final  value 84097.285620
## converged
## # weights:  21
## initial  value 85453.725711
## iter  10 value 84122.931048
## iter  20 value 84097.664432
## iter  30 value 84097.278499
## final  value 84097.248439
## converged
## # weights:  25
## initial  value 85845.016917
## iter  10 value 84246.720879
## iter  20 value 84097.915360
## iter  30 value 84097.245796
## iter  40 value 84097.213743
## iter  40 value 84097.213310
## iter  40 value 84097.213117
## final  value 84097.213117
## converged
## # weights:  29
## initial  value 85868.830473
## iter  10 value 84138.958603
## iter  20 value 84098.121603
## iter  30 value 84097.251149
## final  value 84097.191889
## converged
## # weights:  33
## initial  value 85840.374671
## iter  10 value 84138.847731
## iter  20 value 84098.097676
## iter  30 value 84097.212560
## final  value 84097.169963
## converged
## # weights:  37
## initial  value 85105.418550
## iter  10 value 84107.520547
## iter  20 value 84097.582848
## iter  30 value 84097.166921
```

```
## final  value 84097.148762
## converged
## # weights:  41
## initial  value 85908.179057
## iter  10 value 84107.787005
## iter  20 value 84097.240079
## iter  30 value 84097.139429
## final  value 84097.134090
## converged
```

```r
#Plotting

plot(x= 1:10, y= train_MSE[1:10], type="b", xlab="Iterations", ylab= "Mean Squared Error",
     col="red", main = "Varying Hidden Nodes for Training")
```




```r
plot(x= 1:10, type="b", y= test_MSE[1:10], xlab="Iterations", ylab= "Mean Squared Error",
     col="green", main = "Varying Hidden Nodes for Testing")
```



We see from the plot that the minimum value for mean squared error occurs at 2 hidden layers.

Problem 11.7

Fit a neural network to the spam data of Section 9.1.2, and compare the results to those for the additive model given in that chapter. Compare both the classification performance and interpretability of the final model.

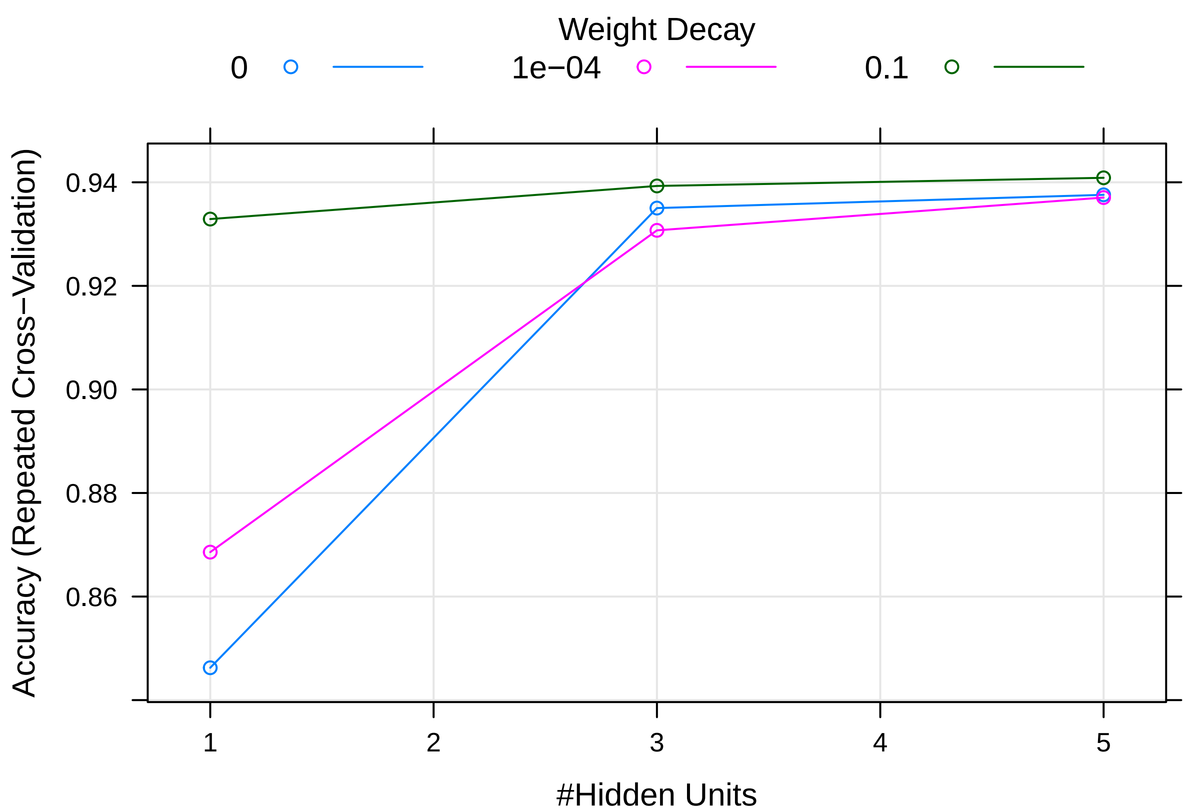
Generating Confusion Matrix for the Neural Network

```
#461 total for confusion matrix

pred_nnet <- predict(nnet_train, spam_test_data[,-58])

confusionMatrix(pred_nnet,spam_test_data$spam)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction email spam
##      email   282   14
##      spam     11  154
##
##                Accuracy : 0.9458
##                  95% CI : (0.921, 0.9646)
```

```
##      No Information Rate : 0.6356
##      P-Value [Acc > NIR] : <2e-16
##
##                    Kappa : 0.8825
##
##   Mcnemar's Test P-Value : 0.6892
##
##              Sensitivity : 0.9625
##              Specificity : 0.9167
##           Pos Pred Value : 0.9527
##           Neg Pred Value : 0.9333
##               Prevalence : 0.6356
##           Detection Rate : 0.6117
##     Detection Prevalence : 0.6421
##        Balanced Accuracy : 0.9396
##
##         'Positive' Class : email
##
```

```r
plot(nnet_train)
```



Section 9.1.2 Confusion Matrix

Table 1: Actual/Predicted Confusion Matrix (9.1.2)

	email	spam
email	269	11
spam	14	167

The confusion matrix meterics from section 9.1.2 are:

Accuracy: 0.9458

Sensitivity: 0.9505

Specificity: 0.9382

Precision (PPV): 0.9607

NPV: 0.9227

Comparing the above confusion matrix meterics with the Neural Network model we see no decrease in accuracy from 0.9458 to 0.9458.

The Sensitivity going from 0.9505 to 0.9625,

The Specificity going from 0.9382 to 0.9167,

The Precision(PPV) going from 0.9607 to 0.9527,

and the NPV going from to 0.9227 to 0.9333.

So using the neural network slightly improved the classification performance but not signficantly.