Homework 3 - Neural Networks and Gradient Descent

Charles Liu (304804942)

5/3/2020

Homework 3 Requirements

There is no separate instruction file. This file is the instructions and you will modify it for your submission.

You will submit two files.

The files you submit will be:

1. `102b_hw3_output_First_Last.Rmd` Take this R Markdown file and make the necessary edits so that it generates the requested output.
2. `102b_hw3_output_First_Last.pdf` OR `102b_hw_3_output_First_Last.html` Your output file. This can be a PDF or an HTML file. This is the primary file that will be graded. **Make sure all requested output is visible in the output file.**

Failure to submit all files will result in an automatic 40 point penalty.

Academic Integrity

Modifying the following statement with your name.

"By including this statement, I, Charles Liu, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students."

If you collaborated verbally with other students, please also include the following line to credit them.

"I did discuss ideas related to the homework with no one. At no point did I show another student my code, nor did I look at another student's code."

For this homework assignment, we will build and train a simple neural network, using the famous "iris" dataset. We will take the four variables `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width` to create a prediction for the species.

We will train the network using gradient descent.

Task 0:

Split the iris data into a training and testing dataset. Scale the data so the numeric variables are all between 0 and 1.

```
# Split between training and testing data
set.seed(1)
n <- dim(iris)[1]
rows <- sample(1:n, 0.8 * n)
train <- iris[rows,]
test <- iris[-rows,]

# Training Data
train$Sepal.Length.s <- train$Sepal.Length/max(train$Sepal.Length)
train$Petal.Length.s <- train$Petal.Length/max(train$Petal.Length)
train$Sepal.Width.s <- train$Sepal.Width/max(train$Sepal.Width)
train$Petal.Width.s <- train$Petal.Width/max(train$Petal.Width)

# Testing Data
test$Sepal.Length.s <- test$Sepal.Length/max(test$Sepal.Length)
test$Petal.Length.s <- test$Petal.Length/max(test$Petal.Length)
test$Sepal.Width.s <- test$Sepal.Width/max(test$Sepal.Width)
test$Petal.Width.s <- test$Petal.Width/max(test$Petal.Width)
```

Setting up our network

Our neural network will have four neurons in the input layer - one for each numeric variable in the dataset. Our output layer will have three outputs - one for each species. There will be a `Setosa`, `Versicolor`, and `Virginica` node. When the neural network is provided 4 input values, it will produce an output where one of the output nodes has a value of 1, and the other two nodes have a value of 0. This is a similar classification strategy we used for the classification of handwriting digits.

I have arbitrarily chosen to have three nodes in our hidden layer.

We will add bias values before applying the activation function at each of our nodes in the hidden and output layers.

See Lecture 3-2.

Task 1:

How many parameters are present in our model? List how many are present in: weight matrix 1, bias values for the hidden layer, weight matrix 2, and bias values for output layer.

Your answer: w(1) = (4 x 3) (12 parameters), b(1) = (3 x 1) (3 parameters), w(2) = (3 x 3) (9 parameters), and b(2) = (3 x 1) (3 parameters).

Task 2:

To express the categories correctly, we need to turn the factor labels in species column into vectors of 0s and 1s. For example, an iris of species *setosa* should be expressed as `1 0 0`. Write some code that will do this.

Hint: you can use `as.integer()` to turn a factor into numbers, and then use a bit of creativity to turn those values into vectors of 1s and 0s.

```
train$speciesNum <- as.integer(train$Species)
test$speciesNum <- as.integer(test$Species)

# Convert Training Data:
train$setosa <- (train$speciesNum == 1)
train$setosa <- as.integer(train$setosa)
train$versicolor <- (train$speciesNum == 2)
train$versicolor <- as.integer(train$versicolor)
train$virginica <- (train$speciesNum == 3)
train$virginica <- as.integer(train$virginica)

# Convert Testing Data:
test$setosa <- (test$speciesNum == 1)
test$setosa <- as.integer(test$setosa)
test$versicolor <- (test$speciesNum == 2)
test$versicolor <- as.integer(test$versicolor)
test$virginica <- (test$speciesNum == 3)
test$virginica <- as.integer(test$virginica)
```

Notation

We will define each matrix of values as follows:

$W^{(1)}$ the weights applied to the input layer.

$B^{(1)}$ are the bias values added before activation in the hidden layer.

$W^{(2)}$ the weights applied to the values coming from the hidden layer.

$B^{(2)}$ are the bias values added before the activation function in the output layer.

J is a matrix of 1s so that the bias values in B can be added to all rows.

Sigmoid Activation function

We will use the sigmoid function as our activation function.

$$f(t) = \frac{1}{1 + e^{-t}}$$

Forward Propagation

$$\underset{N\times 4}{\mathbf{X}}\ \underset{4\times 3}{\mathbf{W}^{(1)}} + \underset{N\times 1}{\mathbf{J}}\ \underset{1\times 3}{\mathbf{B}^{(1)T}} = \underset{N\times 3}{\mathbf{z}^{(2)}}$$

$$f(\underset{N\times 3}{\mathbf{z}^{(2)}}) = \underset{N\times 3}{\mathbf{a}^{(2)}}$$

$$\underset{N\times 3}{\mathbf{a}^{(2)}}\ \underset{3\times 3}{\mathbf{W}^{(2)}} + \underset{N\times 1}{\mathbf{J}}\ \underset{1\times 3}{\mathbf{B}^{(2)T}} = \underset{N\times 3}{\mathbf{z}^{(3)}}$$

$$f(\underset{N\times 3}{\mathbf{z}^{(3)}}) = \underset{N\times 3}{\hat{\mathbf{y}}}$$

Easier Notation without bold face and without the dimensions underneath:

$$XW^{(1)} + JB^{(1)T} = Z^{(2)}$$

$$f(Z^{(2)}) = A^{(2)}$$

$$A^{(2)}W^{(2)} + JB^{(2)T} = Z^{(3)}$$

$$f(Z^{(3)}) = \hat{Y}$$

Task 3:

Express the forward propagation as R code using the training data. For now use random uniform values as temporary starting values for the weights and biases.

```
# Create our layer sizes
input_layer_size <- 4
hidden_layer_size <- 3
output_layer_size <- 3

# Create the Sigmoid function
sigmoid <- function(t) {
    1/(1 + exp(-t))
}

# Create our matrix from training data
X <- as.matrix(train[, 6:9])
Y <- as.matrix(train[, 11:13])

# Initialize our weight matrices with random weights
set.seed(1)
W_1 <- matrix(runif(input_layer_size * hidden_layer_size) - 0.5, nrow =
                input_layer_size)
B1 <- matrix(runif(hidden_layer_size), ncol = 1)
W_2 <- matrix(runif(hidden_layer_size * output_layer_size) - 0.5, nrow =
                hidden_layer_size)
B2 <- matrix(runif(output_layer_size), ncol = 1)

# Using Handwriting Ann MNIST Data example
batch_size <- 120 # arbitrarily chosen
Z_2 <- X %*% W_1
A_2 <- sigmoid(Z_2 + t(B1 %*% rep(1, batch_size)))
Z_3 <- A_2 %*% W_2
Y_hat <- sigmoid(Z_3 + t(B2 %*% rep(1, batch_size)))
```

Back Propagation

The cost function that we will use to evaluate the performance of our neural network will be the squared error cost function:

$$J = 0.5 \sum (y - \hat{y})^2$$

```r
# Create the Cost function
cost <- function(y, y_hat) {
  0.5*sum((y - y_hat)^2)
}

# Cost function value with y and y_hat
cost(Y, Y_hat)
```

```
## [1] 54.66568
```

Task 4: (The hard task)

Find the gradient of the cost function with respect to the parameter matrices.

You will create four partial derivatives, one for each of $(W^{(1)}, B^{(1)}, W^{(2)}, B^{(2)})$.

This is known as back propagation. The value of the cost function (J) ultimately depends on the data (X) and our predictions $(\hat{Y})$. Our predictions $(\hat{Y})$ are just a result of a series of operations as seen above. Thus, when you calculate the derivative of the cost function, you will be applying the chain rule for derivatives as you take the derivative with respect to an early element.

Keep in mind that the derivative of J with respect to a matrix will have the same dimensions as the matrix.

I recommend that you do the work on paper by hand, writing the dimensions underneath each component to make sure they align.

I hope the work done for you in Lecture 3-3 will be helpful as a guide to solving this.

You do not need to show all of your work. You only need to typeset your final answers.

Feel free to discuss working through the steps with each other, but don't share your full solutions to the derivatives with each other or on Campuswire.

Your answer:

$$\frac{\partial J}{\partial W^{(2)}} = -(A)^T * (y - \hat{y}) * f'(Z^{(3)} + B^{(2)})$$

$$\frac{\partial J}{\partial B^{(2)}} = -(1)^T * (y - \hat{y}) * f'(Z^{(3)} + B^{(2)})$$

$$\frac{\partial J}{\partial W^{(1)}} = -(X)^T * (y - \hat{y}) * f'(Z^{(3)} + B^{(2)}) * (W^{(2)})^T * f'(Z^{(2)} + B^{(1)})$$

$$\frac{\partial J}{\partial B^{(1)}} = -(1)^T * (y - \hat{y}) * f'(Z^{(3)} + B^{(2)}) * (W^{(2)})^T * f'(Z^{(2)} + B^{(1)})$$

Task 5:

Turn your partial derivatives into R code. To make sure you have coded it correctly, it is always a good idea to perform numeric gradient checking, which will be your next task.

```r
# Create the Sigmoid Prime function
sigmoidprime <- function(z) {
    exp(-z)/((1 + exp(-z))^2)
}

# Create the delta matrices with our batch size
```

```
delta3 <- (-(Y - Y_hat) * sigmoidprime(Z_3 + t(B2 %*% rep(1, batch_size))))
delta2 <- delta3 %*% t(W_2) * sigmoidprime(Z_2 + t(B1 %*% rep(1, batch_size)))
djdb1 <- rep(1, batch_size) %*% delta2
djdw1 <- t(X) %*% delta2

djdb2 <- rep(1, batch_size) %*% delta3
djdw2 <- t(A_2) %*% delta3
```

Numerical gradient checking

Task 6:

Perform numeric gradient checking. For the purpose of this homework assignment, show your numeric gradient checking for just the $W^{(1)}$ matrix. You should do numeric gradient checking for all elements in your neural network, but for the sake of keeping the length of this assignment manageable, show your code and results for the first weight matrix only.

To perform numeric gradient checking, create an initial set of parameter values for all of the values (all weight matricies and all bias values). Calculate the predicted values based on these initial parameters, and calculate the cost associated with them. Store this 'initial' cost value.

You will then perturb one element in the $W^{(1)}$ matrix by a small value, say 1e-4. You will then recalculate the predicted values and associated cost. The difference between the new value of the cost function and the initial cost gives us an idea of the change in J. Divide that change by the size of the perturbation (1e-4), and we now have an idea of the slope (partial derivative). You'll repeat this for all of the elements in the $W^{(1)}$ matrix.

You've already done a similar task in HW2. Please also see the example code in "neural network example code" available in week 4 on CCLE.

```
# Set up the problem
set.seed(1)
e <- 1e-4
input_layer_size <- 4
output_layer_size <- 3
hidden_layer_size <- 3

# Set up matrices
W_1 <- matrix(runif(input_layer_size * hidden_layer_size) - 0.5, nrow =
                input_layer_size, ncol = hidden_layer_size)
W_2 <- matrix(runif(hidden_layer_size * output_layer_size) - 0.5, nrow =
                hidden_layer_size, ncol = output_layer_size)
B1 <- matrix(runif(hidden_layer_size), ncol = 1)
B2 <- matrix(runif(output_layer_size), ncol = 1)
Z_2 <- X %*% W_1
A_2 <- sigmoid(Z_2 + t(B1 %*% rep(1, 120)))
Z_3 <- A_2 %*% W_2
Y_hat <- sigmoid(Z_3 + t(B2 %*% rep(1, 120)))

# Initialize the numgrad_w_1
numgrad_w_1 <- matrix(0, nrow = input_layer_size, ncol = hidden_layer_size)
elements <- (input_layer_size * hidden_layer_size)
currentcost <- cost(Y, Y_hat)

# Create the for() loop
for (i in 1:elements) {
```

```
    set.seed(1)
    W_1 <- matrix(runif(input_layer_size * hidden_layer_size) - 0.5, nrow =
                    input_layer_size, ncol = hidden_layer_size)
    W_2 <- matrix(runif(hidden_layer_size * output_layer_size) - 0.5, nrow =
                    hidden_layer_size, ncol = output_layer_size)
    B1 <- matrix(runif(hidden_layer_size), ncol = 1)
    B2 <- matrix(runif(output_layer_size), ncol = 1)
    W_1[i] <- W_1[i] + e
    Z_2 <- X %*% W_1
    A_2 <- sigmoid(Z_2 + t(B1 %*% rep(1, 120)))
    Z_3 <- A_2 %*% W_2
    Y_hat <- sigmoid(Z_3 + t(B2 %*% rep(1, 120)))
    numgrad_w_1[i] <- (cost(Y, Y_hat) - currentcost)/e
}

# The matrix from the numgrad_w_1
numgrad_w_1
```

```
##            [,1]       [,2]      [,3]
## [1,] 0.19776155 0.37039655 1.6694495
## [2,] 0.31747135 0.08978447 1.1259662
## [3,] 0.07311966 0.49634684 1.7151075
## [4,] 0.34903529 0.01095857 0.9552443
```

Now check to make sure that the values produced by the numeric gradient check match the values of the gradient as calculated by the partial derivatives which you calculated in the previous task. The match won't be perfect, but should be pretty good.

```
# Create the matrices to check
set.seed(1)
W_1 <- matrix(runif(input_layer_size * hidden_layer_size) - 0.5, nrow =
                input_layer_size, ncol = hidden_layer_size)
W_2 <- matrix(runif(hidden_layer_size * output_layer_size) - 0.5, nrow =
                hidden_layer_size, ncol = output_layer_size)
B1 <- matrix(runif(hidden_layer_size), ncol = 1)
B2 <- matrix(runif(output_layer_size), ncol = 1)
Z_2 <- X %*% W_1
A_2 <- sigmoid(Z_2 + t(B1 %*% rep(1, 120)))
Z_3 <- A_2 %*% W_2
Y_hat <- sigmoid(Z_3 + t(B2 %*% rep(1, 120)))

# Create the matrices for the deltas
delta3 <- (-(Y - Y_hat) * sigmoidprime(Z_3 + t(B2 %*% rep(1, 120))))
delta2 <- delta3 %*% t(W_2) * sigmoidprime(Z_2 + t(B1 %*% rep(1, 120)))
djdw1 <- t(X) %*% delta2

# Check and compare the djdw1 with the numgrad_w_1
djdw1
```

```
##                      [,1]       [,2]      [,3]
## Sepal.Length.s 0.19776304 0.37040043 1.6694340
## Petal.Length.s 0.31747320 0.08978338 1.1259548
## Sepal.Width.s  0.07312017 0.49635497 1.7150940
## Petal.Width.s  0.34903729 0.01095687 0.9552338
```

Gradient Descent

Task 7:

We will now apply the gradient descent algorithm to train our network. This simply involves repeatedly taking steps in the direction opposite of the gradient.

With each iteration, you will calculate the predictions based on the current values of the model parameters. You will also calculate the values of the gradient at the current values. Take a 'step' by subtracting a scalar multiple of the gradient. And repeat.

I will not specify what size scalar multiple you should use, or how many iterations need to be done. Just try things out. A simple way to see if your model is performing 'well' is to print out the predicted values of y-hat and see if they match closely to the actual values.

```
# Create the matrices
set.seed(1)
X <- as.matrix(train[, 6:9])
Y <- as.matrix(train[, 11:13])
W_1 <- matrix(runif(input_layer_size * hidden_layer_size) - 0.5, nrow =
                input_layer_size)
B1 <- matrix(runif(runif(3)), ncol = 1)
W_2 <- matrix(runif(hidden_layer_size * output_layer_size) - 0.5, nrow =
                hidden_layer_size)
B2 <- matrix(runif(runif(3)), ncol = 1)
scalar <- 0.01 # scale it by about 0.01

# Start up the for() loop
for (i in 1:10000) {
    Z_2 <- X %*% W_1
    A_2 <- sigmoid(Z_2 + t(B1 %*% rep(1, 120)))
    Z_3 <- A_2 %*% W_2
    Y_hat <- sigmoid(Z_3 + t(B2 %*% rep(1, 120)))
    cost(Y, Y_hat)
    delta3 <- (-(Y - Y_hat) * sigmoidprime(Z_3 + t(B2 %*% rep(1, 120))))
    djdb2 <- rep(1, 120) %*% delta3
    djdw2 <- t(A_2) %*% delta3
    delta2 <- delta3 %*% t(W_2) * sigmoidprime(Z_2 + t(B1 %*% rep(1, 120)))
    djdb1 <- rep(1, 120) %*% delta2
    djdw1 <- t(X) %*% delta2
    W_1 <- W_1 - scalar * djdw1
    B2 <- B2 - scalar * t(djdb2)
    W_2 <- W_2 - scalar * djdw2
    B1 <- B1 - scalar * t(djdb1)
}
```

Testing our trained model

Now that we have performed gradient descent and have effectively trained our model, it is time to test the performance of our network.

Task 8:

Using the testing data, create predictions for the 30 observations in the test dataset. Print those results.

```
X2 <- as.matrix(test[, 6:9])
Y2 <- as.matrix(test[, 11:13])
```

```
Z2 <- X2 %*% W_1
A2 <- sigmoid(Z2 + t(B1 %*% rep(1, 30)))
Z3 <- A2 %*% W_2
Y2_hat <- sigmoid(Z3 + t(B2 %*% rep(1, 30)))
prediction <- round(Y2_hat)
prediction
```

```
##     setosa versicolor virginica
## 4        1          0         0
## 5        1          0         0
## 8        1          0         0
## 9        1          0         0
## 11       1          0         0
## 16       1          0         0
## 30       1          0         0
## 36       1          0         0
## 46       1          0         0
## 47       1          0         0
## 49       1          0         0
## 52       0          1         0
## 54       0          1         0
## 57       0          0         1
## 62       0          1         0
## 67       0          0         1
## 69       0          0         1
## 72       0          1         0
## 80       0          1         0
## 90       0          1         0
## 96       0          1         0
## 97       0          1         0
## 99       0          1         0
## 103      0          0         1
## 107      0          0         1
## 109      0          0         1
## 117      0          0         1
## 128      0          0         1
## 137      0          0         1
## 139      0          0         1
```

Task 9:

Create a confusion matrix - a table that compares the predictions to the actual values. It will be a 3x3 table, and most values should be along the diagonal. Off-diagonal elements represent errors.

You can create a simple confusion matrix by making some adjustements to the following code

```
vector_of_predicted_class <- as.vector(prediction[, 1:3])
vector_of_true_class <- as.vector(Y2[, 1:3])
results_df <- data.frame(predicted = vector_of_predicted_class, actual =
                           vector_of_true_class)
table(results_df)
```

```
##          actual
## predicted  0  1
##         0 57  3
```

```
##          1  3 27
```

How many errors did your network make? It made 3 errors.

Using package nnet

While instructive, the manual creation of a neural network is seldom done in production environments.

[Install the `nnet` and `NeuralNetTools` packages] Read the documentation for the function `nnet()`. I've created a neural network for predicting the iris species based on the four numeric variables. We use the same training data to train the network. The function `nnet()` is smart enough to recognize that the values in the species column are a factor and will need to expressed in 0s and 1s as we did in our manually created network.

```
library(nnet)
library(NeuralNetTools)
```

```
## Warning: package 'NeuralNetTools' was built under R version 3.6.3
```

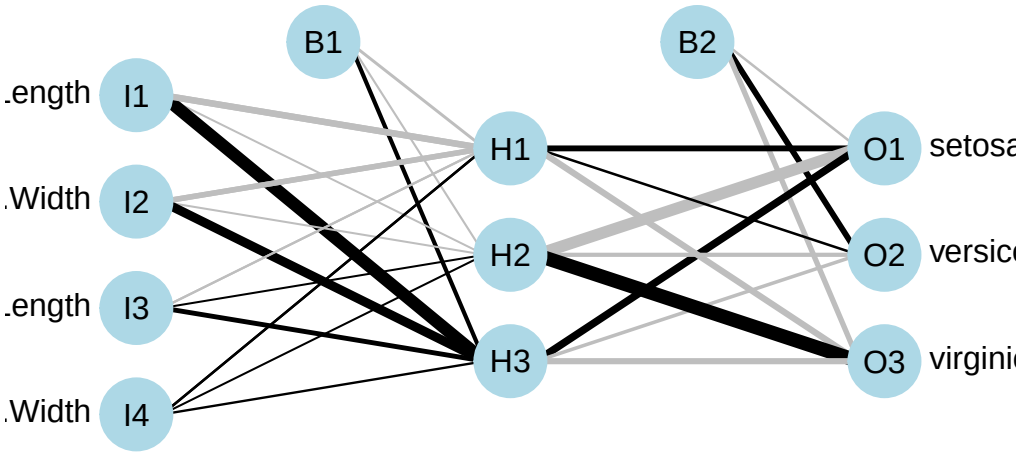
```
set.seed(1)
n <- dim(iris)[1]
rows <- sample(1:n, 0.8*n)
train <- iris[rows,]


irismodel <- nnet(Species ~ Sepal.Length + Sepal.Width + Petal.Length +
                    Petal.Width, size=3, data = train)
```

```
## # weights:  27
## initial  value 144.247567
## iter  10 value 51.531244
## iter  20 value 11.693157
## iter  30 value 4.794318
## iter  40 value 4.644015
## iter  50 value 4.642486
## iter  60 value 4.632611
## iter  70 value 4.625519
## iter  80 value 4.618919
## iter  90 value 4.616588
## iter 100 value 4.612555
## final  value 4.612555
## stopped after 100 iterations
```

Once we have created the network with nnet, we can use the predict function to make predictions for the test data.

```
plotnet(irismodel) # a plot of our network
```




```
results <- max.col(predict(irismodel, iris[-rows,]))
results_df <- data.frame(results, actual = as.numeric(iris[-rows, 5]))
results_df
```

```
##    results actual
## 1        1      1
## 2        1      1
## 3        1      1
## 4        1      1
## 5        1      1
## 6        1      1
## 7        1      1
## 8        1      1
## 9        1      1
## 10       1      1
## 11       1      1
## 12       2      2
## 13       2      2
## 14       2      2
## 15       2      2
## 16       2      2
## 17       2      2
## 18       2      2
## 19       2      2
## 20       2      2
## 21       2      2
```

```
## 22       2      2
## 23       2      2
## 24       3      3
## 25       2      3
## 26       3      3
## 27       3      3
## 28       3      3
## 29       3      3
## 30       2      3
```

```r
table(results_df)
```

```
##         actual
## results  1  2  3
##       1 11  0  0
##       2  0 12  2
##       3  0  0  5
```

```r
# we can see that the predicted probability of each class matches the actual label
```