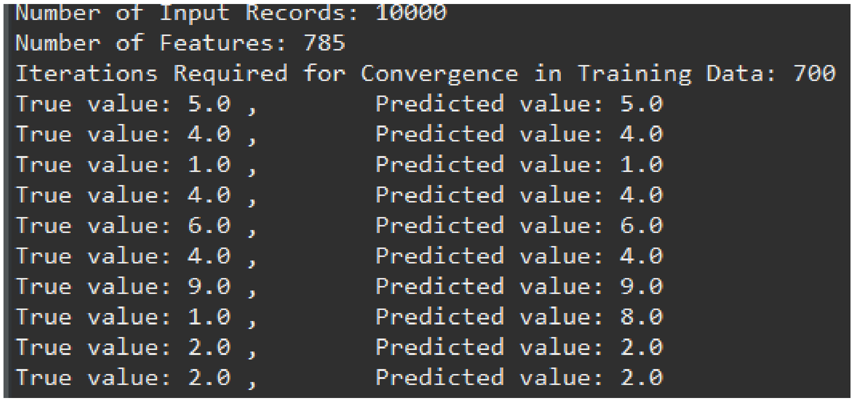
Noah Buchanan
Problem Set 5
AI at 5:25 PM

April 16, 2021

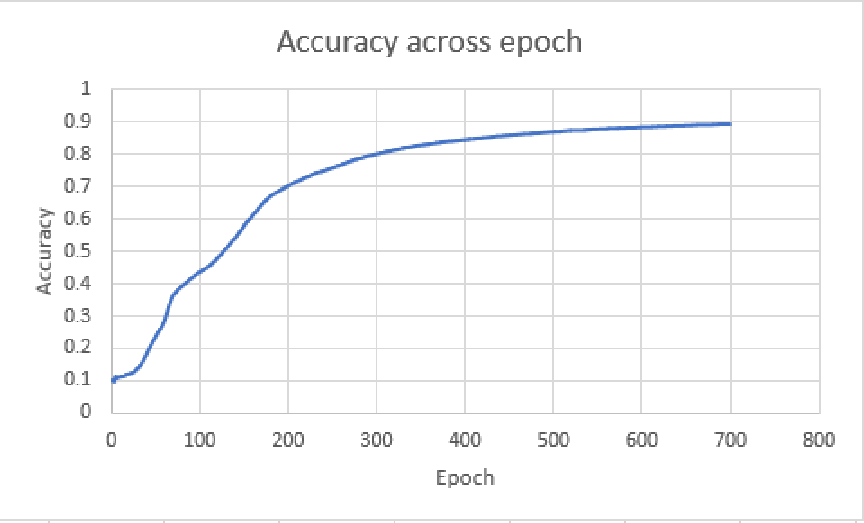
Below are the details of the output reported at the very end of the algorithm.



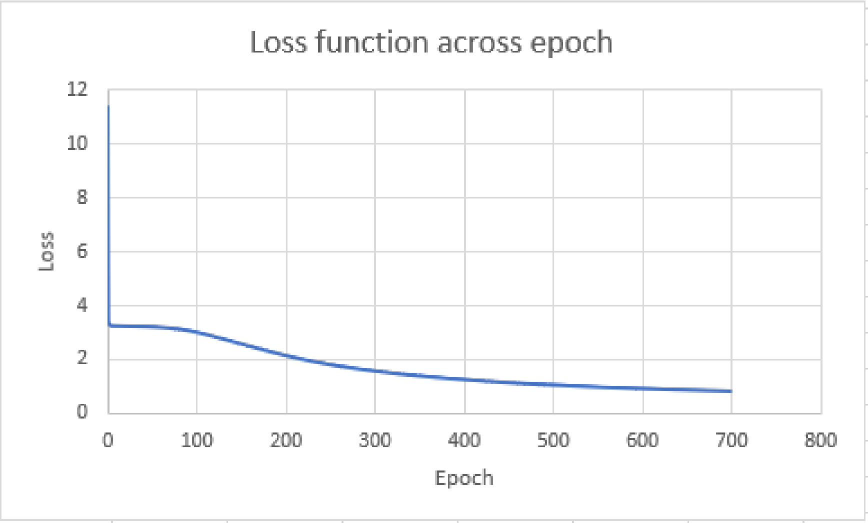
```
Number of Input Records: 10000
Number of Features: 785
Iterations Required for Convergence in Training Data: 700
True value: 5.0 ,        Predicted value: 5.0
True value: 4.0 ,        Predicted value: 4.0
True value: 1.0 ,        Predicted value: 1.0
True value: 4.0 ,        Predicted value: 4.0
True value: 6.0 ,        Predicted value: 6.0
True value: 4.0 ,        Predicted value: 4.0
True value: 9.0 ,        Predicted value: 9.0
True value: 1.0 ,        Predicted value: 8.0
True value: 2.0 ,        Predicted value: 2.0
True value: 2.0 ,        Predicted value: 2.0
```

I believe that it is noteworthy to include that the accuracy could have been improved even further had we not simplified the convergence criteria to 700. I implemented my own convergence criteria that you mentioned in class where 3 consecutive decreases in i'th epoch's accuracy would converge. I ran it from the output weights from the previous 700 epoch run and the accuracy continued to increase, I however do not know to what degree it would increase as the program is quite strenuous and time consuming to run so I will not be finding out solely for the purpose of experimentation and a pending due date on this Problem Set.

1 Accuracy



2 Loss



3 Neural Network source code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

/******************************
Name: Noah Buchanan
Username: ua100
Problem Set: PS5
Due Date: April 16, 2021
******************************/


public class PS5 extends Matrix{

   static double[][] X;
   static double[][] y;
   static double[][] yvector;
   static double[][] w1;
   static double[][] w2;
   static double[][] yhat;
   static double[][] yhatvector;
   static double[][] delta1;
   static double[][] delta2;
   static double[][] H1;
   static double lambda = 3;
   static double learnRate = 0.25;

   public static void main(String[] args) throws IOException{

      X = prependBias(readin(”xdata”));
      yvector = readin(”ydata”);
      y = oneHotEncodeY(yvector);
      w1 = readin(”w1out”);
      w2 = readin(”w2out”);

      GradientDescent();


      try {
         BufferedWriter bw = new BufferedWriter(new FileWriter(”w2out”));

         String line = ””;

         for(int j = 0; j < w2.length; j++) {
```

```
         for(int k = 0; k < w2[0].length; k++) {
            bw.write(w2[j][k]+”,”);
         }
         bw.newLine();
      }
      bw.close();
   } catch (Exception ex) {
      ex.printStackTrace();
   }
}

public static void GradientDescent() throws IOException{

   BufferedWriter bw1 = new BufferedWriter(new FileWriter(”accuracy”));
   BufferedWriter bw2 = new BufferedWriter(new FileWriter(”loss”));

   int downcount = 0;
   double accuracy = 0;
   int epoch = 0;

   while(downcount < 3 && epoch < 700){

      double[][] weightedSum;
      //hidden layer weighted sum and activation function
      weightedSum = multiply(X,transpose(w1));
      for(int j = 0; j < weightedSum.length; j++) {
         for(int k = 0; k < weightedSum[0].length; k++) {
            weightedSum[j][k] =  h(weightedSum[j][k]);
         }
      }

      weightedSum = prependBias(weightedSum);
      H1 = weightedSum;
      //output layer weighted sum and activation function
      weightedSum = multiply(weightedSum,transpose(w2));
      for(int j = 0; j < weightedSum.length; j++) {
         for(int k = 0; k < weightedSum[0].length; k++) {
            weightedSum[j][k] =  h(weightedSum[j][k]);
         }
      }

      yhat = weightedSum;

      double count = 0;
      yhatvector = new double[weightedSum.length][1];
      for(int j = 0; j < yhat.length; j++) {
```

```
    double max = 0;
    int index = 0;
    for(int k = 0; k < yhat[0].length; k++) {
        if(max < yhat[j][k]) {
            max = yhat[j][k];
            index = k;
        }
    }
    yhatvector[j][0] = index+1;
    if(yvector[j][0] == yhatvector[j][0]) {
        count++;
    }
}
if(accuracy > count/yhat.length) {
    downcount++;
} else {
    downcount = 0;
}
accuracy = count/yhat.length;
bw1.write(String.format(”%d , %.3f”,epoch,accuracy));
bw1.newLine();

//loss function calculation

bw2.write(String.format(”%d , %.3f”, epoch, loss()));
bw2.newLine();

//backpropogation................................................
delta2 = delta2();
delta1 = delta1();

double[][] gradientw2 = multiply(transpose(delta2),H1);
double[][] gradientw1 = multiply(transpose(delta1),X);

//*w1 and 2 no bias*
double[][] w2nb = biasMitigation(w2);
double[][] w1nb = biasMitigation(w1);

for(int j = 0; j < gradientw2.length; j++) {
    for(int k = 0; k < gradientw2[0].length; k++) {
        gradientw2[j][k] = gradientw2[j][k] + lambda*w2nb[j][k];
        gradientw2[j][k] = gradientw2[j][k]/y.length;
    }
}

for(int j = 0; j < gradientw1.length; j++) {
        for(int k = 0; k < gradientw1[0].length; k++) {
            gradientw1[j][k] = gradientw1[j][k] + lambda*w1nb[j][k];
            gradientw1[j][k] = gradientw1[j][k]/y.length;
        }
    }

    for(int j = 0; j < gradientw2.length; j++) {
        for(int k = 0; k < gradientw2[0].length; k++) {
            w2[j][k] = w2[j][k] + learnRate*gradientw2[j][k];
        }
    }

    for(int j = 0; j < gradientw1.length; j++) {
        for(int k = 0; k < gradientw1[0].length; k++) {
            w1[j][k] = w1[j][k] + learnRate*gradientw1[j][k];
        }
    }
    epoch++;
}
bw1.close();
bw2.close();

BufferedWriter bw3 = new BufferedWriter(new FileWriter(”w1out”));

for(int j = 0; j < w1.length; j++) {
    for(int k = 0; k < w1[0].length; k++) {
        bw3.write(String.format(”%.3f,”, w1[j][k]));
    }
    bw3.newLine();
}
bw3.close();


BufferedWriter bw4 = new BufferedWriter(new FileWriter(”w2out”));

for(int j = 0; j < w2.length; j++) {
    for(int k = 0; k < w2[0].length; k++) {
        bw4.write(String.format(”%.3f,”, w2[j][k]));
    }
    bw4.newLine();
}
bw4.close();

System.out.printf(”Number of Input Records: %d\n”, X.length);
System.out.printf(”Number of Features: %d\n”, X[0].length);
System.out.printf(”Iterations Required for Convergence in Training Data: %
        for(int l = 0; l < 10; l++) {
            System.out.println("True value: " + yvector[l][0] + " ,\t Predicted val
        }
    }

    public static double loss() {

        //getting regularization value
        double regularization = 0;
        for(int i = 0; i < w1.length; i++) {
            for(int j = 0; j < w1[0].length; j++) {
                regularization += Math.pow(w1[i][j], 2);
            }
        }
        for(int i = 0; i < w2.length; i++) {
            for(int j = 0; j < w2[0].length; j++) {
                regularization += Math.pow(w2[i][j], 2);
            }
        }
        regularization /= ( 2 * yhat.length );
        regularization *= 3;

        double sum = 0;

        for(int i = 0; i < yhat.length; i++) {
            for(int k = 0; k < 10; k++) {
                sum += (  -y[i][k]*Math.log(yhat[i][k])   ) - (
((double)1-y[i][k])*Math.log(1-yhat[i][k])   );
            }
        }
        sum += regularization;
        sum /= yhat.length;

        return sum;
    }

    public static double[][] delta2(){
        double[][] temp = new double[yhat.length][yhat[0].length];

        for(int i = 0; i < yhat.length; i++) {
            for(int j = 0; j < yhat[0].length; j++) {
                temp[i][j] = y[i][j] - yhat[i][j];
            }
        }
        return temp;
```

```
}

public static double[][] delta1(){

   double[][] temp = multiply(delta2,dropFirstColumn(w2));
   double[][] xwT = multiply(X,transpose(biasMitigation(w1)));
   for(int i = 0; i < xwT.length; i++) {
      for(int j = 0; j < xwT[0].length; j++) {
         xwT[i][j] = h(xwT[i][j]) * ((double)1 - h(xwT[i][j]));
      }
   }
   return hadamardProduct(temp,xwT);
}

public static double[][] biasMitigation(double[][] w){
   double[][] temp = new double[w.length][w[0].length];
   for(int i = 0; i < temp.length; i++) {
      for(int j = 0; j < temp[0].length; j++) {
         if(j==0) {
            temp[i][j] = 0;
         } else {
            temp[i][j] = w[i][j];
         }
      }
   }
   return temp;
}

public static double[][] hadamardProduct(double[][] x, double[][] y) {
   double[][] temp = new double[x.length][x[0].length];

   for(int i = 0; i < x.length; i++) {
      for(int j = 0; j < x[0].length; j++) {
         temp[i][j] = x[i][j] * y[i][j];
      }
   }

   return temp;
}

public static double[][] oneHotEncodeY(double[][] y){
   double[][] temp = new double[y.length][10];
   for(int i = 0; i < y.length; i ++) {
      temp[i][(int) (y[i][0]-1)] = 1;
   }
   return temp;
```

```
}

public static double h(double xwT) {
    return 1/(1+Math.exp(-xwT));
}

public static double[][] readin(String filename){

    ArrayList<ArrayList<Double>> data = new ArrayList<>();
    try {

        BufferedReader br = new BufferedReader(new FileReader(filename));
        int row = 0;
        String line = ””;

        while(( line = br.readLine() ) != null) {
            data.add(new ArrayList<>());
            String[] split = line.split(”,”);
            for(int i = 0; i < split.length; i++) {
                data.get(row).add(Double.parseDouble(split[i]));
            }
            row++;
        }
        double[][] matrix = new double[data.size()][data.get(0).size()];
        for(int i = 0; i < data.size(); i++){
            for(int j = 0; j < data.get(0).size(); j++){
                matrix[i][j] = data.get(i).get(j);
            }
        }

        return matrix;
    } catch(Exception ex) {
        ex.printStackTrace();
        return null;
    }
}

public static void printMatrix(double[][] data) {
    for(int i = 0; i < data.length; i++){
        for(int j = 0; j < data[0].length; j++){
            System.out.print(data[i][j]+” , ”);
        }
        System.out.println();
    }
}
```

```
    public static double[][] prependBias(double[][] matrix) {

        double[][] temp = new double[matrix.length][matrix[0].length+1];

        for(int i = 0; i < matrix.length; i++) {
            for(int j = 0; j < matrix[0].length; j++) {
                temp[i][j+1] = matrix[i][j];
            }
        }

        for(int i = 0; i < temp.length; i++) {
            temp[i][0] = 1;
        }

        return temp;
    }
}
```