Tópicos em Avanços Computacionais I

Rede Neural Simples – Java

Joinvile Batista Junior

```
package dl;

import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import org.apache.commons.math3.linear.RealMatrix;
import org.apache.commons.math3.linear.MatrixUtils;

public class NeuralNetwork {
    private final int nodes_nLi, nodes_nLh, nodes_nLo;  // L : Layer
    private final double learning_rate;
    private RealMatrix Wih, Who; // W : Weights between two Layers
```

```
public NeuralNetwork
(int nodes_nLi, int nodes_nLh, int nodes_nLo, double learning_rate) {

        this.nodes_nLi = nodes_nLi;
        this.nodes_nLh = nodes_nLh;
        this.nodes_nLo = nodes_nLo;
        this.learning_rate = learning_rate;

        Wih = generate_randomW(nodes_nLh, nodes_nLi);
        Who = generate_randomW(nodes_nLo, nodes_nLh);
    }
```



```
private RealMatrix generate_randomW(int rows, int columns) {
        RealMatrix randomW = MatrixUtils.createRealMatrix(rows, columns);
        for(int row = 0; row < rows; row++){
            for(int column = 0; column < columns; column++){
                randomW.setEntry(row, column,
                        ThreadLocalRandom.current().nextDouble(-0.5,0.5));
            }
        }
        return randomW;
    }
```

```
public void trainNeuralNetwork(List input_data, List target_data) {

        RealMatrix outputLi = listToVector(input_data).transpose();
        RealMatrix outputLh = applyNeuralLayer(outputLi, Wih);
        RealMatrix outputLo = applyNeuralLayer(outputLh, Who);

        RealMatrix targetLo = listToVector(target_data).transpose();
        RealMatrix errorLo = targetLo.subtract(outputLo);
        Who = updateW(Who, outputLh, outputLo, errorLo);

        RealMatrix errorLh = Who.transpose().multiply(errorLo);
        Wih = updateW(Wih, outputLi, outputLh, errorLh);
}
```



```
  private RealMatrix listToVector(List list){
      int list_size = list.size();
      RealMatrix vector = MatrixUtils.createRealMatrix(1, list_size);
      for(int column = 0; column < list_size; column++){
          vector.setEntry(0, column, (double)list.get(column));
      }
      return vector;
   }
```

```
public RealMatrix applyNeuralNetwork(List input_data){
    RealMatrix outputLi = listToVector(input_data).transpose();
    RealMatrix outputLh = applyNeuralLayer(outputLi, Wih);
    RealMatrix outputLo = applyNeuralLayer(outputLh, Who);
    return outputLo;
}

private RealMatrix applyNeuralLayer
(RealMatrix outputLL, RealMatrix W) {
    return activationFunction(W.multiply(outputLL)); // LL : left Layer
}
```



```
private RealMatrix activationFunction(RealMatrix inputRL) {
    int rows = inputRL.getRowDimension();
    int columns = inputRL.getColumnDimension();
    RealMatrix outputRL = MatrixUtils.createRealMatrix(rows, columns);
    for(int row = 0; row < rows; row++){
        for(int column = 0; column < columns; column++){
            double element_inputRL = inputRL.getEntry(row,column);
            double element_outputRL
                    = 1.0 / (1.0 + Math.exp(-element_inputRL)); // sigmoid
            outputRL.setEntry(row, column, element_outputRL);
        }
    }
    return outputRL;
}
```

```
private RealMatrix updateW
          (RealMatrix W, RealMatrix outputLL,  // LL : left Layer
           RealMatrix outputRL, RealMatrix errorRL) {  // RL : wright Layer
       int delta_rows = errorRL.getRowDimension();
       int delta_columns = errorRL.getColumnDimension();
       RealMatrix deltaW = MatrixUtils.createRealMatrix
              (delta_rows, delta_columns);
       for(int row = 0; row < delta_rows; row++){
          for(int column = 0; column < delta_columns; column++){
             double element_errorRL = errorRL.getEntry(row,column);
             double element_outputRL = outputRL.getEntry(row,column);
             deltaW.setEntry(row, column,
                    element_errorRL * element_outputRL
                       * (1.0 - element_outputRL));
          }
       }
       deltaW = deltaW.multiply(outputLL.transpose())
                    .scalarMultiply(learning_rate);
       return W.add(deltaW);
    }
}
```



```
package dl;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
import org.apache.commons.math3.linear.RealMatrix;

public class App1_Java {

    static final int TRAINING_EXAMPLES_TOTAL = 60000;
    static final int TEST_EXAMPLES_TOTAL = 10000;
    static final int NUMBER_PIXELS = 784;
    static final int DIGITS_TOTAL = 10;
    static final double MAX_PIXEL = 255.0;
    static final double MAX_VALUE = 0.99;
    static final double MIN_VALUE = 0.01;
```

```
static int nodes_nLh = 200;
static double learning_rate = 0.1;
static int epochs_total = 5;

private static ArrayList<ArrayList<Double>> manual_numbers
    = new ArrayList();
private static ArrayList<ArrayList<Double>> target_numbers
    = new ArrayList();
private static ArrayList<Double> manual_one_number
    = new ArrayList();
private static ArrayList<Double> target_one_number
    = new ArrayList();
private static NeuralNetwork neural_network;
```



```
public static void main(String args[]) {
   neural_network = new NeuralNetwork
       (NUMBER_PIXELS, nodes_nLh, DIGITS_TOTAL, learning_rate);
   trainNeuralNetWork();
   testNeuralNetWork();
   evaluateNeuralNetWork();
}
```

```
private static void trainNeuralNetWork() {
    String training_file_name = "data/mnist_train.csv";
    File training_file = new File(training_file_name);
    for (int epoch = 0; epoch < epochs_total; epoch++) {
        try {
            Scanner inputStream = new Scanner(training_file);
            String[][] number_pixels = new String[1][NUMBER_PIXELS + 1];
            while (inputStream.hasNext()) {
                for (int example = 0;
                     example < TRAINING_EXAMPLES_TOTAL; example++) {
                    for (int digit = 0; digit < DIGITS_TOTAL; digit++) {
                        target_one_number.add(MIN_VALUE);
                    }
                    number_pixels[0] = inputStream.next().split(",");
                    target_one_number.set(Integer.parseInt
                         (number_pixels[0][0]), MAX_VALUE);
```



```
                    for (int pixel = 1; pixel < (NUMBER_PIXELS + 1); pixel++) {
                        manual_one_number.add(((Double.parseDouble
                            (number_pixels[0][pixel])) / MAX_PIXEL
                             * MAX_VALUE) + MIN_VALUE);
                    }
                    neural_network.trainNeuralNetwork
                         (manual_one_number, target_one_number);
                    manual_one_number = new ArrayList();
                    target_one_number = new ArrayList();
                }
            }
            inputStream.close();
        } catch (FileNotFoundException exception) {
            exception.printStackTrace();
        }
    }
}
```

```
private static void testNeuralNetWork() {
    String test_file_name = "data/mnist_test.csv";
    File test_file = new File(test_file_name);
    try {
        Scanner inputStream = new Scanner(test_file);
        String[][] one_character = new String[1][NUMBER_PIXELS + 1];
        while (inputStream.hasNext()) {
            for (int example = 0;
                 example < TEST_EXAMPLES_TOTAL; example++) {
                for (int digit = 0; digit < DIGITS_TOTAL; digit++) {
                    target_one_number.add(MIN_VALUE);
                }
                one_character[0] = inputStream.next().split(",");
                target_one_number.set(Integer.parseInt
                     (one_character[0][0]), MAX_VALUE);
                target_numbers.add(target_one_number);
```



```
                for (int pixel = 1; pixel < (NUMBER_PIXELS + 1); pixel++) {
                    manual_one_number.add(((Double.parseDouble
                            (one_character[0][pixel])) / (MAX_PIXEL
                             * MAX_VALUE)) + MIN_VALUE);
                }
                manual_numbers.add(manual_one_number);
                manual_one_number = new ArrayList();
                target_one_number = new ArrayList();
            }
        }
        inputStream.close();
    } catch (FileNotFoundException exception) {
        exception.printStackTrace();
    }
}
```

```
private static void evaluateNeuralNetWork() {
   int[] score_board = new int[TEST_EXAMPLES_TOTAL];
   for(int example = 0;
        example < TEST_EXAMPLES_TOTAL; example++) {
      RealMatrix result = neural_network.applyNeuralNetwork
          (manual_numbers.get(example));
      int targets_max_number =
          maxArrayValue(target_numbers.get(example));
      int result_max_number = maxValue(result);
      if (targets_max_number == result_max_number)
          score_board[example] = 1;
      else score_board[example] = 0;
   }
   double score_board_values_sum = 0;
   for(int example = 0;
        example < TEST_EXAMPLES_TOTAL; example++){
      score_board_values_sum += score_board[example];
   }
   System.out.println("Performance: "
        + score_board_values_sum/TEST_EXAMPLES_TOTAL);
}
```



```
private static int maxValue(RealMatrix matrix){
      int max_value = 0;
      int rows_n = matrix.getRowDimension();
      for(int row_n = 0; row_n < rows_n; row_n++){
         if(matrix.getEntry(row_n,0) > matrix.getEntry(max_value, 0)){
            max_value = row_n;
         }
      }
      return max_value;
}
```

```
    private static int maxArrayValue(ArrayList<Double> vector){
            int max_value = 0;
            int vector_size = vector.size();
            for (int element_n = 0; element_n < vector_size; element_n++) {
                if (vector.get(element_n) > vector.get(max_value)){
                    max_value = element_n;
                }
            }
            return max_value;
    }

}
```