Tópicos em Avanços Computacionais I

Rede Neural Simples – ND4J

Joinvile Batista Junior

```
package dl;

import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.factory.Nd4j;
import org.nd4j.linalg.ops.transforms.Transforms;

public class NeuralNetwork {

    private final int nodes_nLi, nodes_nLh, nodes_nLo;
    private final double learning_rate;
    private INDArray Wih, Who;
```

```
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 INDArray generate_randomW(int rows, int columns) {
    INDArray randomW = Nd4j.create(rows,columns);
    for(int row = 0; row < rows; row++){
        for(int column = 0; column < columns; column++) {
            randomW.putScalar(row,column,
                ThreadLocalRandom.current().nextDouble(-0.5,0.5));
        }
    }
    return randomW;
}
```

```
public void trainNeuralNetwork(List input_list, List target_list) {
    INDArray outputLi = toArray2D(input_list).transpose();
    INDArray outputLh = applyNeuralLayer(outputLi, Wih);
    INDArray outputLo = applyNeuralLayer(outputLh, Who);

    INDArray targetLo = toArray2D(target_list).transpose();
    INDArray errorLo = targetLo.sub(outputLo);
    Who = updateW(Who, outputLh, outputLo, errorLo);

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



```
private INDArray toArray2D(List list) {
    int list_size = list.size();
    INDArray array2D = Nd4j.create(1,list_size);
    for(int column = 0; column < list_size; column++){
        array2D.putScalar(0,column, (double) list.get(column));
    }
    return array2D;
}
```

```
public INDArray applyNeuralNetwork(List input_list) {
    INDArray outputLi = toArray2D(input_list).transpose();
    INDArray outputLh = applyNeuralLayer(outputLi, Wih);
    INDArray outputLo = applyNeuralLayer(outputLh, Who);
    return outputLo;
}

private INDArray applyNeuralLayer(INDArray outputLL, INDArray W) {
    return Transforms.sigmoid(W.mmul(outputLL)); // LL : left Layer
}
```



```
private INDArray updateW(INDArray W, INDArray outputLL,
        INDArray outputRL, INDArray errorRL) {
    int delta_rows = errorRL.rows();
    int delta_columns = errorRL.columns();
    INDArray deltaW = Nd4j.create(delta_rows, delta_columns);
    for(int row = 0; row < delta_rows; row++) {
        for(int column = 0; column < delta_columns; column++) {
            double element_errorRL = errorRL.getDouble(row, column);
            double element_outputRL = outputRL.getDouble(row, column);
            deltaW.putScalar(row, column,
              element_errorRL * element_outputRL
                *(1.0 - element_outputRL));
        }
    }
    deltaW = deltaW.mmul(outputLL.transpose()).mul(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.nd4j.linalg.api.ndarray.INDArray;

public class App1_ND4J {

    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);
    manual_numbers = new ArrayList();
    target_numbers = new ArrayList();
    manual_one_number = new ArrayList();
    target_one_number = new ArrayList();
    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++) {
           INDArray prediction = neural_network.applyNeuralNetwork
                 (manual_numbers.get(example));
           int targets_max_number = indexMaxArrayValue
                 (target_numbers.get(example));
           int prediction_max_number = indexMaxValue(prediction);
           if(targets_max_number == prediction_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 indexMaxValue(INDArray matrix) {
        int max_value = 0;
        int rows = matrix.rows();
        for(int row = 0; row < rows; row++){
           if(matrix.getDouble(row,0)
              > matrix.getDouble(max_value, 0)) max_value = row;
        }
        return max_value;
    }
```

```
    private static int indexMaxArrayValue(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;
    }

}
```