```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public class NeuralNet implements NeuralNetInterface {
    //hyper-parameters
    private boolean binary = true;
    private int numInputs = 2;
    private int numHidden = 4;
    private int numOutputs = 1;
    private double learningRate = 0.2;
    private double momentum = 0.9;
    private double a = 0;
    private double b = 1;
    private double initCeiling = 0.5;
    private double initFloor = -0.5;
    private static double errorThreshold = 0.05;

    //layers
    private double[] inputLayer = new double[numInputs + 1];
  //one extra bias node
    private double[] hiddenLayer = new double[numHidden + 1]
;
    private double[] outputLayer = new double[numOutputs];


    //weights
    private double[][] w1 = new double[numInputs + 1][
numHidden];
    private double[][] w2 = new double[numHidden + 1][
numOutputs];

```

```
36      //back-propagation update arrays
37      private double[] deltaOutput = new double[numOutputs];
38      private double[] deltaHidden = new double[numHidden];
39
40      private double[][] deltaW1 = new double[numInputs + 1][
    numHidden];
41      private double[][] deltaW2 = new double[numHidden + 1][
    numOutputs];
42
43      //error data
44      private double[] totalError = new double[numOutputs];
45      private double[] singleError = new double[numOutputs];
46
47      private List<String> errorList = new LinkedList<>();
48
49      //training set
50      private double[][] trainX; //one bias node
51      private double[][] trainY;
52
53      public NeuralNet(int numInputs, int numHidden, int
    numOutputs, double learningRate, double momentum, double a,
    double b) {
54          this.numInputs = numInputs;
55          this.numHidden = numHidden;
56          this.numOutputs = numOutputs;
57          this.learningRate = learningRate;
58          this.momentum = momentum;
59          this.a = a;
60          this.b = b;
61      }
62
63      public NeuralNet() {
64      }
65
66      public void initializeTrainSet() {
67          if (binary) {
68              trainX = new double[][]{{0, 0}, {0, 1}, {1, 0},
    {1, 1}};
69              trainY = new double[][]{{0}, {1}, {1}, {0}};
```

```
70          } else {
71              trainX = new double[][]{{-1, -1}, {-1, 1}, {1,
   -1}, {1, 1}};
72              trainY = new double[][]{{-1}, {1}, {1}, {-1}};
73          }
74      }
75
76      //bipolar sigmoid
77      @Override
78      public double sigmoid(double x) {
79          return 2 / (1 + Math.exp(-x)) - 1;
80      }
81
82      @Override
83      public double customSigmoid(double x) {
84          if (!binary) {
85              b = 1;
86              a = -1;
87          }
88          return (b - a) / (1 + Math.exp(-x)) + a;
89      }
90
91      @Override
92      public void initializeWeights() {
93          for (int i = 0; i < numInputs + 1; i++) {
94              for (int j = 0; j < numHidden; j++) { //no
   weights for the bias node
95                  double r = new Random().nextDouble();
96                  w1[i][j] = initFloor + (r * (initCeiling -
   initFloor));
97                  deltaW1[i][j] = 0.0;
98              }
99          }
100
101         for (int j = 0; j < numHidden + 1; j++) {
102             for (int k = 0; k < numOutputs; k++) {
103                 double r = new Random().nextDouble();
104                 w2[j][k] = initFloor + (r * (initCeiling -
   initFloor));
```

```
105                 deltaW2[j][k] = 0.0;
106             }
107         }
108     }
109 
110     @Override
111     public void zeroWeights() {
112         //w1, w2 entries are automatically assigned default
    zero by compiler
113     }
114 
115     private void initializeLayers(double[] sample) {
116         for (int i = 0; i < numInputs; i++) {
117             inputLayer[i] = sample[i];
118         }
119         inputLayer[numInputs] = 1;
120         hiddenLayer[numHidden] = 1;
121     }
122 
123     private void forwardPropagation(double[] sample) {
124         initializeLayers(sample);
125         for (int j = 0; j < numHidden; j++) {
126             hiddenLayer[j] = 0;
127             for (int i = 0; i < numInputs + 1; i++) {
128                 hiddenLayer[j] += w1[i][j] * inputLayer[i];
129             }
130             hiddenLayer[j] = customSigmoid(hiddenLayer[j]);
131         }
132 
133         for (int k = 0; k < numOutputs; k++) {
134             outputLayer[k] = 0;
135             for (int j = 0; j < numHidden + 1; j++) {
136                 outputLayer[k] += w2[j][k] * hiddenLayer[j]
    ;
137             }
138             outputLayer[k] = customSigmoid(outputLayer[k]);
139 
140         }
141         //System.out.println(singleError[0]);
```

```
142     }
143 
144     private void backPropagation() {
145         //compute deltaOutput[]
146         for (int k = 0; k < numOutputs; k++) {
147             deltaOutput[k] = 0;
148             deltaOutput[k] = binary ? singleError[k] *
    outputLayer[k] * (1 - outputLayer[k]) :
149                     singleError[k] * (outputLayer[k] + 1) *
     0.5 * (1 - outputLayer[k]);
150         }
151 
152         //update w2
153         for (int k = 0; k < numOutputs; k++) {
154             for (int j = 0; j < numHidden + 1; j++) {
155                 deltaW2[j][k] = momentum * deltaW2[j][k] +
    learningRate * deltaOutput[k] * hiddenLayer[j];
156                 w2[j][k] += deltaW2[j][k];
157             }
158         }
159 
160         //Compute deltaHidden
161         for (int j = 0; j < numHidden; j++) {
162             deltaHidden[j] = 0;
163             for (int k = 0; k < numOutputs; k++) {
164                 deltaHidden[j] += w2[j][k] * deltaOutput[k]
    ;
165             }
166             deltaHidden[j] = binary ? deltaHidden[j] *
    hiddenLayer[j] * (1 - hiddenLayer[j]) :
167                     deltaHidden[j] * (hiddenLayer[j] + 1) *
     0.5 * (1 - hiddenLayer[j]);
168         }
169 
170 
171         //Update w1
172         for (int j = 0; j < numHidden; j++) {
173             for (int i = 0; i < numInputs + 1; i++) {
174                 deltaW1[i][j] = momentum * deltaW1[i][j]
```

```
175                         + learningRate * deltaHidden[j] *
    inputLayer[i];
176                 w1[i][j] += deltaW1[i][j];
177             }
178         }
179
180     }
181
182     public int train() {
183         errorList.clear();
184         int epoch = 0;
185
186         do {
187             for (int k = 0; k < numOutputs; k++) {
188                 totalError[k] = 0;
189             }
190             int numSamples = trainX.length;
191             for (int i = 0; i < numSamples; i++) {
192                 double[] sample = trainX[i];
193                 forwardPropagation(sample);
194                 for (int k = 0; k < numOutputs; k++) {
195                     singleError[k] = trainY[i][k] -
    outputLayer[k];
196                     totalError[k] += Math.pow(singleError[k
    ], 2);
197                 }
198                 backPropagation();
199             }
200
201             for (int k = 0; k < numOutputs; k++) {
202                 totalError[k] /= 2;
203                 System.out.println("Total error for output
    number " + (k + 1) + ": " + totalError[k]);
204             }
205             errorList.add(Double.toString(totalError[0]));
206             epoch++;
207         } while (totalError[0] > errorThreshold);
208         System.out.println("This trial epoch " + epoch + "\
    n");
```

```
209             return epoch;
210         }
211 
212         @Override
213         public double outputFor(double[] X) {
214             forwardPropagation(X);
215             return outputLayer[0];
216         }
217 
218         @Override
219         public double train(double[] X, double argValue) {
220             forwardPropagation(X);
221             return argValue - outputLayer[0];
222         }
223 
224         @Override
225         public void save(File argFile) {
226             try {
227                 StringBuilder builder = new StringBuilder();
228                 for (int i = 0; i < w1.length; i++) {
229                     for (int j = 0; j < w1[0].length; j++) {
230                         builder.append(w1[i][j] + " ");
231                     }
232                     builder.append("\n");
233                 }
234                 builder.append("\n");
235                 for (int i = 0; i < w2.length; i++) {
236                     for (int j = 0; j < w2[0].length; j++) {
237                         builder.append(w2[i][j] + " ");
238                     }
239                     builder.append("\n");
240                 }
241                 Files.write(argFile.toPath(), builder.toString(
    ).getBytes());
242             } catch (IOException e) {
243                 e.printStackTrace();
244             }
245         }
246 
```

```
247     @Override
248     public void load(String argFileName) throws IOException
     {
249         Scanner sc = new Scanner(new BufferedReader(new
    FileReader("./weights.txt")));
250         double[][] w1 = new double[numInputs + 1][numHidden
    ];
251         double[][] w2 = new double[numHidden + 1][
    numOutputs];
252         boolean readingW1 = true;
253         int lineIndex = 0;
254         while (sc.hasNextLine()) {
255             if (readingW1) {
256                 String[] line = sc.nextLine().trim().split(
    " ");
257                 if (line[0].length() == 0) {
258                     readingW1 = false;
259                     lineIndex = 0;
260                     continue;
261                 }
262                 //System.out.println(line[0]);
263                 for (int j = 0; j < line.length; j++) {
264                     w1[lineIndex][j] = Double.parseDouble(
    line[j]);
265                 }
266                 lineIndex++;
267             } else {
268                 String[] line = sc.nextLine().trim().split(
    " ");
269                 if (line[0].length() == 0) {
270                     break;
271                 }
272                 //System.out.println(line[0]);
273                 for (int j = 0; j < line.length; j++) {
274                     w2[lineIndex][j] = Double.parseDouble(
    line[j]);
275                 }
276                 lineIndex++;
277             }
```

```
278         }
279     }
280 
281     public void saveError() {
282         try {
283             Files.write(Paths.get("./trainError.txt"),
    errorList);
284         } catch (IOException e) {
285             e.printStackTrace();
286         }
287     }
288 }
289 
```