```java
package classes;

import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import jdk.nashorn.internal.runtime.regexp.joni.ast.QuantifierNode;

/**
 * created by lgcaobianco on 2018-05-21
 */

public class MLP3 {
    private Double[][] W1, W1Anterior, W2, W2Anterior, deltaCamada1;
    private List<Double[]> matrizInputs, matrizOperacao, vetorEntrada;
    private Double[][] I1;
    private Double I2 = 0.0, Y2;
    private Double deltaCamada2 = 0.0;
    private double taxaAprendizagem = 0.1;
    private Double[][] Y1;
    private int qtdEntradas;
    private int qtdNeuronios;
    private Double alfa = 0.8;

    // O construtor irá efetuar as operações essenciais para o funcionamento dos
    // métodos.
    public MLP3(int qtdEntradas, int qtdNeuronios) {
        this.qtdEntradas = qtdEntradas;
        this.qtdNeuronios = qtdNeuronios;

        LeitorPontosEntrada leitor = new LeitorPontosEntrada(
                "/home/lgcaobianco/repositorios/epc-rna/epc6/src/base/conjunto-treinamento", ".csv");
        this.matrizInputs = leitor.extrairPontos();
        ajustarMatrizTreinamento(qtdEntradas);
        this.vetorEntrada = leitor.extrairPontos();
        imprimirQualquer(matrizInputs);
        leitor = null; // para leitor se tornar candidato ao garbage collector

        Random random = new Random();

        W2Anterior = new Double[1][qtdNeuronios + 1];
        W2 = new Double[1][qtdNeuronios + 1];

```

```
44          deltaCamada1 = new Double[qtdNeuronios][1];
45          I1 = new Double[qtdNeuronios][1];
46          Y1 = new Double[qtdNeuronios + 1][1];
47          W1Anterior = new Double[qtdNeuronios][qtdEntradas
    + 2];
48          W1 = new Double[qtdNeuronios][qtdEntradas + 2];
49
50          // sorteia W1
51          for (int i = 0; i < W1.length; i++) {
52              for (int j = 0; j < W1[i].length; j++) {
53                  W1[i][j] = (random.nextDouble()/100);
54                  W1Anterior[i][j] = 0.0;
55              }
56          }
57
58
59          // sorteia W2
60          for (int i = 0; i < W2[0].length; i++) {
61              W2[0][i] = (random.nextDouble()/100);
62              W2Anterior[0][i] = 0.0;
63          }
64
65          System.out.println("W1");
66          imprimirQualquer(W1);
67          System.out.println("W2");
68          imprimirQualquer(W2);
69
70      }
71
72      public List<Double[]> getMatrizInputs() {
73          return matrizInputs;
74      }
75
76      public void setMatrizInputs(List<Double[]>
    matrizInputs) {
77          this.matrizInputs = matrizInputs;
78      }
79
80      public List<Double[]> getMatrizOperacao() {
81          return matrizOperacao;
82      }
83
84      public void setMatrizOperacao(List<Double[]>
    matrizOperacao) {
85          this.matrizOperacao = matrizOperacao;
86      }
87
88      public Double[][] getI1() {
89          return I1;
90      }
```

```
 91
 92      public void setI1(Double[][] i1) {
 93          I1 = i1;
 94      }
 95
 96      public Double[][] getW1() {
 97          return W1;
 98      }
 99
100      public Double[][] getW2() {
101          return W2;
102      }
103
104      public Double[][] getY1() {
105          return Y1;
106      }
107
108      public void obterI1(int linhaMatrizInput) {
109          double soma = 0.0;
110          for (int i = 0; i < W1.length; i++) {
111              for (int j = 0; j < W1[i].length; j++) {
112                  soma += (W1[i][j] * matrizInputs.get(
    linhaMatrizInput)[j]);
113              }
114              I1[i][0] = soma;
115              soma = 0.0;
116          }
117      }
118
119      public void obterY1() {
120          Y1[0][0] = -1.0;
121          for (int i = 0; i < I1.length; i++) {
122              Y1[i + 1][0] = 0.5 + 0.5 * Math.tanh((I1[i][0
    ]) / 2);
123          }
124      }
125
126      public void obterI2() {
127          I2 = 0.0;
128          for (int i = 0; i < W2[0].length; i++) {
129              I2 += Y1[i][0] * W2[0][i];
130          }
131      }
132
133      public void obterY2() {
134          Y2 = 0.0;
135          Y2 = 0.5 + 0.5 * Math.tanh(I2 / 2);
136      }
137
138      public void obterDeltaCamada2(int linhaMatrizInput) {
```

```
139         // A derivada de g(Ij) pode ser expressa como f(x
    ) * (1 - f(x))!
140         this.deltaCamada2 = (matrizInputs.get(
    linhaMatrizInput)[matrizInputs.get(0).length - 1] - Y2) *
     (Y2 * (1 - Y2));
141     }
142 
143     public void ajustarPesosCamada2() {
144         Double[][] aux = new Double[1][W2[0].length];
145 
146         for (int i = 0; i < W2[0].length; i++) {
147             aux[0][i] = W2[0][i];
148             W2[0][i] = W2[0][i] + (taxaAprendizagem *
    deltaCamada2 * Y1[i][0]) + (alfa * (W2[0][i] - W2Anterior
    [0][i]));
149         }
150 
151         for (int i = 0; i < W2[0].length; i++) {
152             W2Anterior[0][i] = aux[0][i];
153         }
154     }
155 
156     public void obterDeltaCamada1() {
157         for (int i = 0; i < deltaCamada1.length; i++) {
158             deltaCamada1[i][0] = deltaCamada2 * W2[0][i]
    * (Y1[i][0] * (1 - Y1[i][0]));
159         }
160 
161     }
162 
163     public void ajustarPesosCamada1(int linhaInputMatriz)
     {
164         Double[][] aux = new Double[W1.length][W1[0].
    length];
165         for (int i = 0; i < W1.length; i++) {
166             for (int j = 0; j < W1[i].length; j++) {
167                 aux[i][j] = W1[i][j];
168                 W1[i][j] = W1[i][j] + (taxaAprendizagem *
     deltaCamada1[i][0] * matrizInputs.get(linhaInputMatriz)[
    j])
169                         + (alfa * (W1[i][j] - W1Anterior[
    i][j]));
170             }
171         }
172 
173         for (int i = 0; i < W1.length; i++) {
174             for (int j = 0; j < W1[i].length; j++) {
175                 W1Anterior[i][j] = aux[i][j];
176             }
177         }
```

```
178     }
179 
180     public double calcularEk(int linhaMatrizEntrada) {
181         double erro = 0.0;
182         erro = Math.pow((matrizInputs.get(
    linhaMatrizEntrada)[qtdEntradas] - Y2), 2);
183         return erro / 2;
184     }
185 
186     public double calcularEm() {
187         double erroTotal = 0.0;
188         for (int i = 0; i < matrizInputs.size(); i++) {
189             erroTotal += calcularEk(i);
190         }
191         double valorRetorno = erroTotal / matrizInputs.
    size();
192         return valorRetorno;
193     }
194 
195     public void forwardPropagation(int linhaMatrizInput)
    {
196         this.obterI1(linhaMatrizInput);
197         this.obterY1();
198         this.obterI2();
199         this.obterY2();
200     }
201 
202     public void backwardPropagation(int linhaMatrizInput)
     {
203         this.obterDeltaCamada2(linhaMatrizInput);
204         this.ajustarPesosCamada2();
205         this.obterDeltaCamada1();
206         this.ajustarPesosCamada1(linhaMatrizInput);
207 
208     }
209 
210     public void ajustarMatrizTreinamento(int
    quantidadeEntradas) {
211         List<Double[]> matrizAuxiliar = new LinkedList<>(
    );
212         int tamanhoConjuntoDados = matrizInputs.size();
213 
214         for (int i = 0; i < (tamanhoConjuntoDados -
    quantidadeEntradas); i++) {
215             Double[] teste = inverterPosicoes(i,
    quantidadeEntradas);
216             matrizAuxiliar.add(i, teste);
217         }
218         this.setMatrizInputs(matrizAuxiliar);
219     }
```

```
220
221     public Double[] inverterPosicoes(int linhaInicial,
    int quantidadeEntradas) {
222         Double[] vetorAuxiliar = new Double[
    quantidadeEntradas + 2];
223         int auxiliar = quantidadeEntradas;
224         vetorAuxiliar[quantidadeEntradas + 1] = this.
    matrizInputs.get(linhaInicial + quantidadeEntradas)[0];
225         vetorAuxiliar[0] = -1.0;
226         for (int i = 0; i < quantidadeEntradas; i++) {
227             vetorAuxiliar[auxiliar] = this.matrizInputs.
    get(i + linhaInicial)[0];
228             auxiliar--;
229         }
230
231         return vetorAuxiliar;
232     }
233
234     public void imprimirQualquer(List<Double[]> objeto) {
235         System.out
236                 .println("O objeto tem dimensoes: " +
    objeto.size() + " linhas e " + objeto.get(0).length + "
    colunas");
237         for (int i = 0; i < objeto.size(); i++) {
238             for (int j = 0; j < objeto.get(i).length; j++
    ) {
239                 System.out.print(objeto.get(i)[j] + " ");
240             }
241             System.out.println();
242         }
243         System.out.println("\n\n");
244     }
245
246     public void imprimirQualquer(Double[][] objeto) {
247         System.out.println("O objeto tem dimensoes: " +
    objeto.length + " linhas e " + objeto[0].length + "
    colunas");
248         for (int i = 0; i < objeto.length; i++) {
249             for (int j = 0; j < objeto[i].length; j++) {
250                 System.out.print(objeto[i][j] + " ");
251             }
252             System.out.println();
253         }
254         System.out.println("\n\n");
255     }
256
257     public void obterNovaAmostra(int posicaoAmostra) {
258         for (int i = 0; i < I1.length; i++) {
259             I1[i][0] = 0.0;
260         }
```

```
261         for (int i = 0; i < W1.length; i++) {
262             I1[i][0] += -1 * W1[i][0];
263             for (int j = 1; j < W1[i].length; j++) {
264                 I1[i][0] += vetorEntrada.get(
    posicaoAmostra - j)[0] * W1[i][j];
265                 // System.out.println("I1[" + i + "] +=
    VetorEntrada(" + (posicaoAmostra - j -
266                 // 1) + ") * " + W1[i][j]);
267             }
268         }
269 
270         obterY1();
271         obterI2();
272         obterY2();
273         System.out.println(Y2);
274         Double[] resultadoRede = new Double[1];
275         resultadoRede[0] = Y2;
276         vetorEntrada.add(resultadoRede);
277 
278     }
279 
280 }
281 
```

```java
package classes;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * created by lgcaobianco on 2018-05-20
 */

public class LeitorPontosEntrada {
    private String nomeArquivo;
    private String formato;
    private String separadorValor;

    private String getNomeArquivo() {
        return nomeArquivo;
    }

    private String getFormato() {
        return formato;
    }

    private String getSeparadorValor() {
        return separadorValor;
    }

    public LeitorPontosEntrada(String nomeArquivo, String formato) {
        this.nomeArquivo = nomeArquivo;
        this.formato = formato;
        switch (formato) {
        case ".csv":
            this.separadorValor = ",";
            break;
        case ".txt":
            this.separadorValor = " ";
            break;
        default:
            System.out.println("Formato ainda não suportado");
            System.exit(1);
            break;
        }
    }

    public List<Double[]> extrairPontos() {
```

```
49          List<Double[]> matrizPontos = new ArrayList<Double
    []>();
50          String linhaLida = "";
51          BufferedReader stream = null;
52          try {
53              stream = new BufferedReader(new FileReader(
    getNomeArquivo() + getFormato()));
54              while ((linhaLida = stream.readLine()) != null
    ) {
55                  String[] temporario = linhaLida.split(
    getSeparadorValor());
56                  Double[] numerosSeparados = new Double[
    temporario.length];
57                  for (int i = 0; i < temporario.length; i++
    ) {
58                      numerosSeparados[i] = Double.
    parseDouble(temporario[i]);
59                  }
60                  matrizPontos.add(numerosSeparados);
61              }
62          } catch (IOException e) {
63              e.printStackTrace();
64              System.exit(1);
65          } finally {
66              if (stream != null) {
67                  try {
68                      stream.close();
69                  } catch (IOException e) {
70                      e.printStackTrace();
71                      System.exit(1);
72                  }
73              }
74          }
75          return matrizPontos;
76      }
77  }
78
```