```
/* questo file contiene le funzioni che permettono la gestione della rete
   neurale. In particolare qui si implementa la fase di apprendimento e di
   addestramento. */

package neurofuzzy;
import org.joone.engine.*;
import java.awt.GridLayout;
import java.awt.*;
import org.joone.io.*;
import java.io.*;
import java.beans.*;
import org.joone.net.NeuralNet;
import org.joone.engine.learning.TeachingSynapse;
import org.joone.net.NeuralNetLoader;
import javax.swing.*;
import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

public class neurale extends JFrame implements NeuralNetListener {

    JProgressBar progress = new JProgressBar(0, 5000);
    Thread esecutore;
    int avanza = 0;
    NeuralNet nnet = new NeuralNet();
    // creazione dei tre strati

    LinearLayer input = new LinearLayer();
    SigmoidLayer hidden = new SigmoidLayer();
    SigmoidLayer output = new SigmoidLayer();

    //definizione del componente trainer

    // riferimento all'oggetto monitor da parte dei vari strati

    TeachingSynapse trainer = new TeachingSynapse();



    public neurale() {

        super();
        setTitle("Addestramento rete in corso...");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        progress.setValue(0);
        progress.setStringPainted(true);
        panel.add(progress);
        setContentPane(panel);



    }

   public void run() {


     while(avanza <= 5000){

         progress.setValue(avanza);

         try {
```

```
            Thread.sleep(1000);

        } catch (InterruptedException e) {}

        avanza += 500;
    }


}


public void inizializza(String uri, String uri2, int num){


    // settaggio dimensioni dei strati

    input.setRows(11);
    hidden.setRows(30);
    output.setRows(1);

    // settaggio nomi degli strati

    input.setLayerName("strato.input");
    hidden.setLayerName("strato.hidden");
    output.setLayerName("strato.output");

    // creazione due tipi di sinapsi

    FullSynapse sinapsi_IH = new FullSynapse(); /* input -> hidden */
    FullSynapse sinapsi_HO = new FullSynapse(); /* hidden -> output */

    // connessione strato input con strato nascosto

    input.addOutputSynapse(sinapsi_IH);
    hidden.addInputSynapse(sinapsi_IH);

    // connessione strato nascosto con strato output

    hidden.addOutputSynapse(sinapsi_HO);
    output.addInputSynapse(sinapsi_HO);


    //creazione sinapsi di input

    FileInputSynapse inputStream = new FileInputSynapse();
    inputStream.setFileName(uri);
    inputStream.setAdvancedColumnSelector("1-11");
    input.addInputSynapse(inputStream);

    //collegamento file risultati con sinapsi di uscita

    FileOutputSynapse fileoutput = new FileOutputSynapse();
    fileoutput.setFileName("risultati.xml");
    output.addOutputSynapse(fileoutput);

    /* Definiamo il file contenente i valori desiderati */

    FileInputSynapse samples = new FileInputSynapse();
    samples.setFileName(uri);

    /* I valori di output desiderati sono nella terza colonna */
```

```
    samples.setAdvancedColumnSelector("12");

    /* Ora attacchiamo il componente di input al trainer */

    trainer.setDesired(samples);
    output.addOutputSynapse(trainer);


    nnet.addLayer(input, NeuralNet.INPUT_LAYER);
    nnet.addLayer(hidden, NeuralNet.HIDDEN_LAYER);
    nnet.addLayer(output, NeuralNet.OUTPUT_LAYER);

    //settaggio parametri monitor

    Monitor monitor = nnet.getMonitor();
    monitor.setLearningRate(0.7);
    monitor.setMomentum(0.5);
    monitor.addNeuralNetListener(this);
    monitor.setTrainingPatterns(num); /* num. di righe nel file di input */
    monitor.setTotCicles(1000); /* Num. totale di cicli di addestramento */
    monitor.setLearning(true);


    //avvio della rete

    //nnet.start();
    //nnet.getMonitor().Go();   /* Partenza! */
    interrogate(uri2);




}

private void interrogate(String name) {


    // neuralNet is an instance of NeuralNet

    Monitor monitor=nnet.getMonitor();
    monitor.setTotCicles(1000);
    monitor.setLearning(true);
    FileOutputSynapse output=new FileOutputSynapse();

    // set the output synapse to write the output of the net

    output.setFileName(name);

    // inject the input and get the output

    if(nnet!=null) {
        nnet.addOutputSynapse(output);
        nnet.start();
        monitor.Go();
        nnet.join();
    }
}

public void salvataggio(String nome_file, neurale nne){

    //Monitor monitor = nnet.getMonitor();
    //monitor.setTotCicles(1);
    //monitor.setLearning(false);
```

```
     /*FileOutputStream stream = new FileOutputStream(nome_file);
    ObjectOutputStream out = new ObjectOutputStream(stream);
    out.writeObject(input);
    out.writeObject(hidden);
    out.writeObject(output);
    out.writeObject(trainer);
    out.close();*/


}

public void caricamento(String nome_file) throws FileNotFoundException,
        IOException, ClassNotFoundException {

    FileInputStream stream = new FileInputStream(nome_file);
    ObjectInputStream inp = new ObjectInputStream(stream);
    Layer input = (Layer)inp.readObject();
    Layer hidden = (Layer)inp.readObject();
    Layer output = (Layer)inp.readObject();
    TeachingSynapse trainer = (TeachingSynapse)inp.readObject();
    Monitor monitor = input.getMonitor();
    monitor.addNeuralNetListener(this);


    input.start();
    hidden.start();
    output.start();
    monitor.Go();



}


public void netStarted(NeuralNetEvent neuralNetEvent) {

    //JOptionPane.showMessageDialog(null, "Addestramento rete in corso...");


}

public void cicleTerminated(NeuralNetEvent neuralNetEvent) {

       Monitor mon = (Monitor)neuralNetEvent.getSource();
       long c = mon.getCurrentCicle();
       long cl = c / 100;

        /*Vogliamo visualizzare l'errore ogni 500 cicli */

       if ((cl * 100) == c){


           System.out.println(""+c + " cicli rimasti - Errore = "+
                              mon.getGlobalError());


       }

}

public void netStopped(NeuralNetEvent neuralNetEvent) {
```

```
            JOptionPane.showMessageDialog(null, "Rete neurale stoppata!!!");


        }

        public void errorChanged(NeuralNetEvent neuralNetEvent) {


        }
        public void netStoppedError(NeuralNetEvent neuralNetEvent, String string) {



        }



}
```