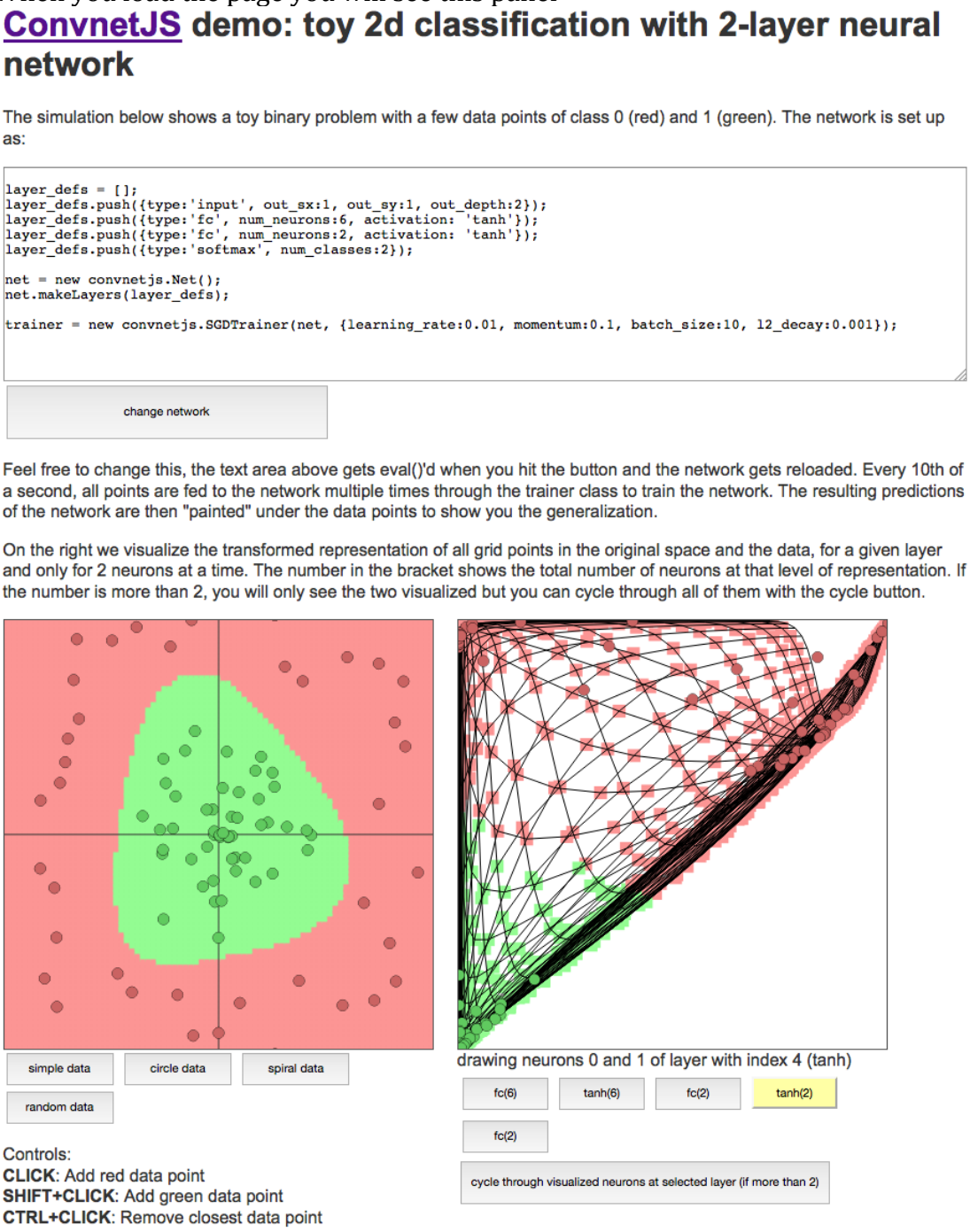
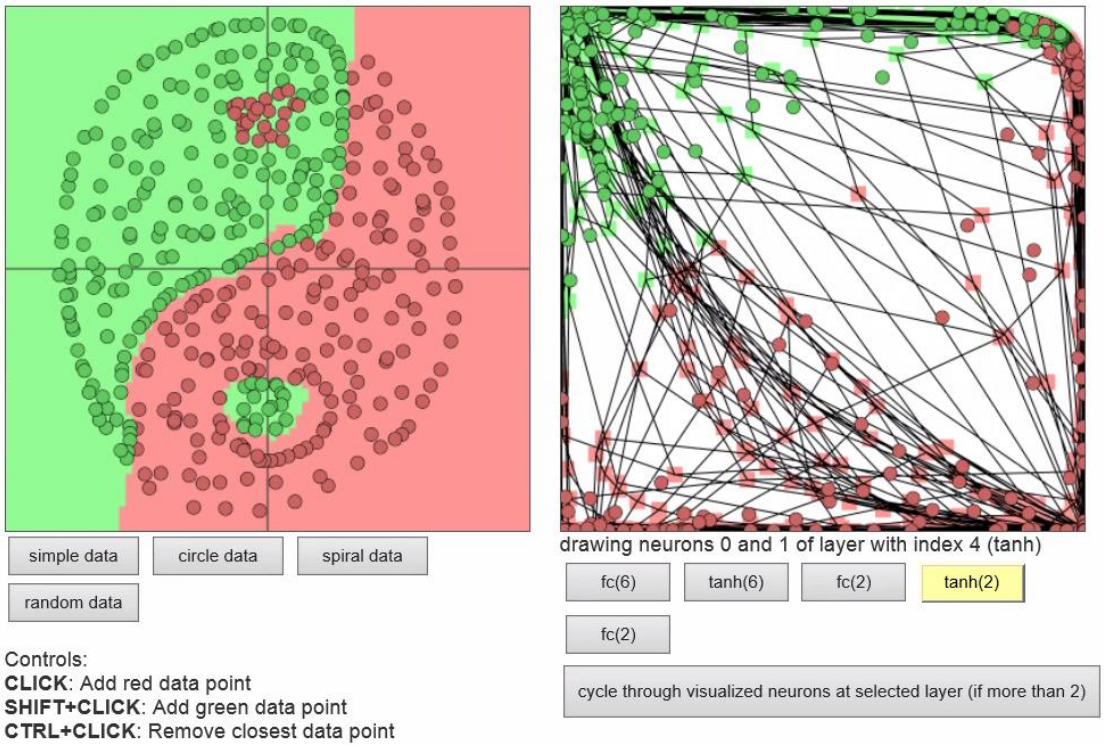
CSC3423 – Biocomputing
Practical5: Neural networks

The aim of this practical is to familiarise yourself with the workings of a Multi-Layer Perceptron (MLP), the most popular type of neural network. To do that you will use a interactive javascript-based MLP implementation.

1. Go to http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html
2. When you load the page you will see this panel



3. The left canvas represents two important aspects: The points represent the two-dimensional data being fed to the NN with the color of the point representing its class. The background colour represents the behaviour of the NN. That is, any coordinate with green background means that that point would be predicted as green. Any coordinate with red background means a red prediction.
4. The right panel shows the values of the neurons in different layers of the NN for specific instances.
5. You have the choice of selecting from a few pre-defined data definitions or write your own. If you generate your own data, the network will automatically adapt to it.
6. The text panel at the top of the page represents the topology of the MLP. You can change the number of layers and the number of neurons in each layer. If you change the topology you need to click on the "change network" button to refresh the network. The description of the syntax of the network definition is here: http://cs.stanford.edu/people/karpathy/convnetjs/docs.html
7. Try different network topologies of increasing difficulty with the spiral problem and see how the network error gets slower and slower.
8. Now rather than using a predefined dataset, try to generate your own data. For instance, does this sound familiar? (if you have read the coursework specification, it should).



Other java NN software packages

There are many different publicly available NN java-based software packages. For instance, Neuroph (http://neuroph.sourceforge.net) is a very comprehensive Java NN library, with implementations of many different types of NN. This program has both a GUI interface and, most importantly for the coursework, a API. This is a simple java example to set up a MLP using this library:

```
package org.neuroph.samples;

import java.util.Arrays;
import org.neuroph.core.NeuralNetwork;
import org.neuroph.nnet.MultiLayerPerceptron;
import org.neuroph.core.data.DataSet;
import org.neuroph.core.data.DataSetRow;
import org.neuroph.util.TransferFunctionType;

/**
* This sample shows how to create, train, save and load simple Multi Layer Perceptron
*/
public class XorMultiLayerPerceptronSample {

    public static void main(String[] args) {

        // create training set (logical XOR function)
        DataSet trainingSet = new DataSet(2, 1);
        trainingSet.addRow(new DataSetRow(new double[]{0, 0}, new double[]{0}));
        trainingSet.addRow(new DataSetRow(new double[]{0, 1}, new double[]{1}));
        trainingSet.addRow(new DataSetRow(new double[]{1, 0}, new double[]{1}));
        trainingSet.addRow(new DataSetRow(new double[]{1, 1}, new double[]{0}));

        // create multi layer perceptron
        MultiLayerPerceptron myMlPerceptron = new
                        MultiLayerPerceptron(TransferFunctionType.TANH, 2, 3, 1);
        // learn the training set
        myMlPerceptron.learn(trainingSet);

        // test perceptron
        System.out.println("Testing trained neural network");
        testNeuralNetwork(myMlPerceptron, trainingSet);

        // save trained neural network
        myMlPerceptron.save("myMlPerceptron.nnet");

        // load saved neural network
        NeuralNetwork loadedMlPerceptron =
                        NeuralNetwork.createFromFile("myMlPerceptron.nnet");

        // test loaded neural network
        System.out.println("Testing loaded neural network");
        testNeuralNetwork(loadedMlPerceptron, trainingSet);

    }

    public static void testNeuralNetwork(NeuralNetwork nnet, DataSet testSet) {

        for(DataSetRow dataRow : testSet.getRows()) {

            nnet.setInput(dataRow.getInput());
            nnet.calculate();
            double[ ] networkOutput = nnet.getOutput();
            System.out.print("Input: " + Arrays.toString(dataRow.getInput()) );
            System.out.println(" Output: " + Arrays.toString(networkOutput) );

        }

    }
}
```

Important aspects from the example:
- Construction of the training set. The first few lines of the main function construct the DataSet object that holds the training data for the MLP. In this case the dataset is simply the truth table of the XOR function. In the case of the coursework, you would need to create the bridge from one set of dataset-holding objects (that of the coursework framework) to the other one (the one from Neuroph). As you can see, the input has two dimensions and the output has one.
- Definition of the MLP structure. The call to the constructor of the MultiLayerPerceptron class specifies the activation function used (TanH in this case) and the number and size of the layers of the MLP. 2 neurons in the input layer (that is, 2 input variables), three neurons in the hidden layer and 1 neuron in the output layer. The list of available activation functions are here: http://neuroph.sourceforge.net/javadoc/org/neuroph/util/TransferFunctionType.html
- The line myMlPerceptron.learn(trainingSet) would train the MLP using the Backpropagation algorithm
- The function testNeuralNetwork tells you how to generate predictions using this MLP, which would need to be interfaced with the coursework's framework.

To build and run this example you need the Neuroph jar files, which are available at http://neuroph.sourceforge.net/download.html. You can build it from the command line or from Eclipse, in a similar way to the examples in previous practicals.