**CSC3423 – Biocomputing**

**Practical4: Neural Networks**

The aim of this practical is to familiarise yourself with the working of the multi-layer perceptron (MLP) algorithm that you have been provided for coursework 2.

Specifically the objectives of this practical are:

1. Learning how to run the MLP code for machine learning
2. Understanding the different parts of the provided code and the neuroph open source library
3. Starting to think about how to generate and extract the performance descriptors for the experiments with the MLP
4. Starting to think about how to optimise the MLP for the provided task

**Getting and running the code.**

The code for the coursework will be in Blackboard (Learning Materials > Assessment > Coursework 2 code).

Before we can run the code, Control.java has to be edited to select the GP version of the code:

public static Classifier generateSubsolution(InstanceSet trainingSet) {

//GAwrapper wrapper = new GAwrapper(trainingSet);

//return wrapper.generateClassifier();

//GPwrapper wrapper = new GPwrapper(trainingSet);

//return wrapper.generateClassifier();

//PSOwrapper wrapper = new PSOwrapper(trainingSet);

//return wrapper.generateClassifier();

**return new ClassifierMLP(trainingSet);**

}

The following panel shows how to call (and the output it produces) the MLP part of the code from the linux command line (the $ just indicates the prompt, don’t type it).

If you run this code from e.g. Eclipse you will need to tell the IDE to load the jar files associated to the Neuroph (<http://neuroph.sourceforge.net/>) neural networks open source library (neuroph-core-2.93.jar and slf4j-api-1.7.5.jar).

The program receives two command line argument which are the names of the training and test files of the provided machine learning dataset. Please note that the code (like any NN) depends on pseudo-random numbers, and hence the output it generates may be different when you run it.

$ java -cp neuroph-core-2.93.jar:slf4j-api-1.7.5.jar:. Control training.arff test.arff

Relation name TAO\_grid

Attribute name x

Attribute name y

Attribute name class

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Classifier of iteration 0. Accuracy 89.81%, coverage 100.00%

Iteration 0, removed 1698 instances, instances left 0

Overall stats at iteration 0. Accuracy 89.81%, error rate 10.19%, not classified 0.00%

Final classifier

cl0:Weight 3.8811561241210373

Weight 1.1943336365076422

Weight 8.278160017452999

Weight -0.6895066313255444

Weight 1.1722880089601135

Weight 0.053619019362927296

Weight -5.599004441150174

Weight -55.67919203671148

Weight -0.5277973295309598

Weight -1.3501545714784018

Weight -1.2275797714620182

Weight 0.13428041979191543

Weight 3.9783285950398346

Weight 1.195443893199547

Weight -8.61792539759364

Weight 9.417222179497324

Weight -4.405142875140498

Weight -7.236600925781419

Weight 2.5491191475485016

Weight 10.868323664240457

Weight -4.910676389404868

Stats on test data

Accuracy 91.05%, error rate 8.95%, not classified 0.00%

Total time: 1.254

**Explanation of the code**

The following lines are just a reproduction (for convenience) of the description of the code in the coursework specification:

The MLP solution has a single class, ClassifierMLP.java. This class has four major functions: createInstance and createDataset are wrappers used to convert the data structures that hold the training data into the format that Neuroph requires.

The ClassifierMLP (constructor) function sets up and trains the network.

* Line 45 sets up the topology of the network: the number of layers and number of neurons per layer. The first argument is the identifier of the activation function used through all the nodes of the network. Afterwards there is a variable list of numbers. The number of elements in this list corresponds to the number of layers in the network. Each number specifies the actual number of neurons in that layer. The numbers for the first and last layer have to be fixed to 2 and 1, respectively. The numbers of the (if any) hidden layers in between can be changed.
* Lines 46 and 47 set up the parameters of the learning process. These parameters are (1) the learning rate, which is the coefficient multiplied to the weight increments during the backpropagation process and (2) the number of iterations of back-propagation.
* Line 48 performs the training process.

classifyInstance is the function that performs predictions of new instances. It queries the instance to specify the values for the network’s input layer, runs the whole network, gets the output and converts this output into a predicted class label. Like in the GP case, a single network predicts the whole dataset, so the training process would finish after one iteration of classifier generation.

**How to measure performance & extract performance measures**

As explained in the coursework specification, there are a variety of performance measures for these experiments. The primary performance measure is the accuracy of the classifier on the test data, which is computed by the framework so you don’t have to do anything. However, from the running of the bio-inspired algorithm (NN in this case) we can also extract other performance measures: how many iterations of the back-propagation process were needed to generate a good classifier? What was the run-time of the algorithm?

**What to do with the code for the coursework?**

You can experiment with:

* The topology of the network: number of hidden layers and number of neurons per hidden layer
* Activation functions

(<http://neuroph.sourceforge.net/javadoc/org/neuroph/util/TransferFunctionType.html>)

* Parameters of the training process: learning rate and number of iterations of the backpropagation process. One caveat about this. The Neuroph package can decide to stop the learning process early and not complete this number of iterations. You can query the library for how many iterations were actually run. Please see getCurrentIteration() and hasReachedStopCondition() of <http://neuroph.sourceforge.net/javadoc/org/neuroph/core/learning/IterativeLearning.html>

You should be systematic. Don’t randomly change many parameters or network topology at once, because then you will not be able to determine what change was the one helping the most in improving the algorithm.