GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

Using Encog as a MLP

Overview

Encog is a machine learning framework that supports a wide variety of technologies, including multi-layer perceptrons (MLPs), radial basis networks (RBNs) and support vector machines (SVMs). The open source framework has been cited in circa 1000 academic publications and is available both on Moodle and from https://www.heatonresearch.com/encog/.

In this lab we will create a MLP using Encog and get it to classify the same data that we used in labs 3 – 6. Most of the work in using Encog is in preparing the data for use by the framework and then deciding on a technology and network topology.

Download the Encog JAR archive from Moodle, add it to your CLASSPATH / Module Path and then create a new class called ***EncogRunner***. We will start with the **game** example, so ensure that you add the arrays called ***data*** and ***expected*** to the class.

Step 1: Declare a Network Topology

Create a three layer neural network as shown below. The number of nodes in each layer is depicted in red. Change these details to match the input vector and output categories for the different examples in labs 3 – 6.

```
BasicNetwork network = new BasicNetwork();
network.addLayer(new BasicLayer(null, true, 4));
network.addLayer(new BasicLayer(new ActivationSigmoid(), true, 2));
network.addLayer(new BasicLayer(new ActivationSigmoid(), false, 4));
network.getStructure().finalizeStructure();
network.reset();
```

We can add additional hidden layers if necessary to create a neural network of arbitrary complexity. Encog support a variety of activation functions including *ActivationStep*, *ActivationTANH*, *ActivationSoftMax* and *ActivationGaussian*.

Step 2: Create the Training Data Set

The simplest way for us to present the training data to Encog is to re-use the existing arrays of double values from labs 3 – 6. In practice, it is more user-friendly to provide the data in CSV format, as this can easily be created from an Excel spreadsheet of data. Encog supports this using the *CSVDataSource* class.

```
MLDataSet trainingSet = new BasicMLDataSet(data, expected);
```

Step 3: Train the Neural Network

Training the neural network involves iterating through the training data for *n* number of epochs until the error level falls below a user-specified threshold. The lower the threshold, the longer

the network will take to train. In this example, we'll set the threshold to 0.09 and use a backpropagation trainer to adjust the weights until $Y(p)$ and $Y_d(p)$ start to converge.

```
ResilientPropagation train = new ResilientPropagation(network, trainingSet);
double minError = 0.09; //Change and see the effect on the result... :)
int epoch = 1;
do {
  train.iteration();
  epoch++;
} while (train.getError() > minError);
train.finishTraining();
```

Step 4: Test the NN

To test the neural network, compare the actual and expected outputs from the training set after the training has completed.

```
for(MLDataPair pair: trainingSet ) {
  MLData output = network.compute(pair.getInput());
  System.out.println(pair.getInput().getData(0) + ","
    + pair.getInput().getData(1)
    + ", Y=" + (int)Math.round(output.getData(0)) //Round the result
    + ", Yd=" + (int) pair.getIdeal().getData(0));
}
```

Step 5: Shutdown the NN

The final step is to shutdown the neural network correctly:

```
Encog.getInstance().shutdown();
```

Exercises

- Change the network ***topology*** to accommodate the different data sets for labs 2-6. Examine the results and pay particular attention to the time it takes to train the network.

- Alter the variable ***minError*** to a lower value, e.g. 0.01 and examine the impact that this has on the training time. You can monitor the falling error level by adding the following statement inside the ***while*** loop:

```
System.out.println("Epoch #" + epoch + " Error:" + train.getError());
```

- Change the number of nodes in the hidden layer to ***under-fit*** and ***over-fit*** the network. Run the programme again and examine the effect of this change on the overall error level and the time that it takes to train the network.

- Add an ***extra layer*** to the neural network with an arbitrary number of nodes (start with a small number) and monitor the effect that this has on the training time and accuracy. A network with two hidden layers should be capable of learning a discontinuous function. Each additional layer increases the computational complexity of the training algorithm exponentially.