COM00007H

Introduction to Neural Computing and Applications

Practical 1

The Perceptron

Section A

The MATLAB Neural Network toolbox provides a set of functions to implement a Perceptron (and many other sorts of network!) The documentation included with MATLAB explains how to use each of these functions, but also explains many of the neural network principles that you need to understand in order to use them correctly. In particular, you should look at the documentation in the 'Getting Started' guide, the section on the neuron model and network architectures, and the section on Perceptrons. Don't worry if not all of it is immediately clear – ask questions now, and the exercises will reinforce the ideas.

The objective of this section is to create a Perceptron. There are a number of steps we need to go through. First, we need to create the MATLAB network object that is our Perceptron, using the `perceptron` function. We also need to have the data in the right format – columns corresponding to individual data items, and rows correponding to variables within those data items. Then, we need to train the perceptron, using the `train()` function. Finally, we can see how the trained perceptron responds to new data.

To create the Perceptron, then, use the following command:

```
>> net = perceptron;
```

Once you have created `net`, you can examine its contents by typing `net` at the command prompt. What you see are the fields making up this object. To examine particular fields, for example the initial weights and bias, type the [object name].[field name]{item name} etc:

```
>> w = net.iw{1,1}, b = net.b{1}

w =

     []


b =

     Empty matrix: 0-by-1
```

You can see the intial weights and bias are undefined. To train the Peceptron, we use the `train` function (this is very general, may be used with many different types of network objects and may take different parameters.).

```
>> net = train(net,P,T);
```

where P is a matrix with columns as data items and rows as variables (i.e. the first two rows of *data2d.mat*), and T is the corresponding target or output data (the third row of *data2d.mat transformed so that it is {0,1} data and not {-1,1} data*).

Notice that the trained network is returned by the function. Calling the function without assigning the result to a variable loses the trained network.

When you call `train`, a window is displayed that shows you the structure of the network you are training, and allows you to track the progress of training (click on the 'performance' button). In this case training is very rapid and there is not much to see.

When training has completed, the weights will have changed, for instance :

```
>> w = net.iw{1,1}, b = net.b{1}

w =

   -1.3679   -0.9308


b =

    1

>>
```

To see what the effect of this on the data is, we can provide data to the network.

```
>> a = net(P);
```

The variable `a` contains the output for each of the columns in the dataset. You should try this, and compare the output with T, the target data.

- How do the results (and weights) compare with what you obtained in Section A?
- Plot the data points, using different symbols for each class. (treat row 1 of the data as an x value, row 2 as a y value).
- On the same plot, add the classifier boundary (choose values of 'x', and use the weights to calculate corresponding values of 'y', then plot a line between these points.)
- The file *data2dns.mat* contains similar data, but this time non separable. How well does the Perceptron perform on this data?

To start training again, use `net = init(net)` to re-initialise the datastructures.

Section B

1. Implement the Perceptron training algorithm (as given below, or in any textbook). You may start from the skeleton given in the MATLAB file *myperceptron.m* in the VLE, but feel free to write your own implementation.

```
initialise weights
repeat {
    select input $\mathbf{x}(t)$ and corresponding output $d(t)$
    calculate $y(t)$
    if $y(t) = d(t)$
        $\mathbf{w}(t+1) = \mathbf{w}(t)$ // do nothing
    else If $y = +1$, should be $-1$
        $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta.\mathbf{x}(t)$
    else // $y = -1$, should be $+1$
        $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta.\mathbf{x}(t)$
} until all x correctly classified
```

2. Using the linearly separable test data from the file *data2d.mat* (which contains 50 observations in columns, and 3 rows containing 2 variables and a binary class indicator), confirm that the algorithm does indeed converge to a classifier capable of distinguishing the two classes. (to check this, use the weights to calculate a class for each item and check they are the same as the target classes.)

3. Now repeat the training with the non-linearly separable test data from *data2dns.mat*. How close does the Perceptron come to a perfect classifier?

4. What happens when you change the learning rate parameter?

5. What happens when you leave out the bias? (by forcing it to zero at each iteration)