Machine Learning CMPU 395

Assignment 2: Multi-Layer Perceptrons

1 Task

The task is to analyze how a Multi-Layer Perceptron classifies data by studying the decision boundaries. You will use the `mlp` class from the book to do the actual learning and classification. You need to write code for generating suitable random data and for plotting the data and the resulting decision boundary.

Minimum requirement is that you produce a plot of a non-linear decision boundary. For highest grade you should explore what happens for different parameter settings, and in particular you should show a situation where overfitting has occured.

2 Software Environment

Like in Assignment 1, you will work in Python and use the `numpy` and `pylab` packages. You will also need the MLP implementation from Stephen Marsland's textbook, which can be downloaded from the books webpage or from the course webpage.

Use the examples in the book to understand how to use the `mlp` package. You will primarily use the methods `mlptrain` (for training the net) and `mlpfwd` (for classification of data).

3 Data Generation

In order to visualize the decision boundaries graphically we will restrict ourselves to two-dimensional data, i.e. points in the plane. We can use the function `random.randn` to generate random points with a normal distribution. This is suitable for our task as we can build up more complex distributions by concatenating sample sets from multiple normal distributions.

`random.randn` takes the number of rows and columns as parameters, like this:

```
classA = numpy.random.randn(50, 2)
```

This will create an array with 50 rows and 2 columns, i.e. 50 two-dimensional datapoints.

`random.randn` always generates numbers with a mean of zero and a standard deviation of one. To place the data where *you* want them and with the deviation (spread) *you* want, you need to multiply with an appropriate scaling factor and to add appropriate offsets. Experiment with this so that you can place your data where you want them. Make use of the same kind of plotting as you used earlier:

```
pylab.plot(classA[:,0], classA[:,1], '.')
pylab.show()
```

The `'.'` at the end tells plot to plot points instead of connecting lines, which does not make much sense here.

Use the function `concatenate` from `numpy` to construct a dataset where the points come from two different normal distributions. You can now use the MLP to classify if a point comes from the first or the second distribution.

For training you must also supply a *target* array. This should be an array with one column and the same number of rows as your data array. It should contain the values zero or one depending on which normal distribution the point comes from. Use the functions `zeros` and `ones` and perform the same concatenation here to create the appropriate target array.

4 Plotting the Decision Boundary

Plotting the decision boundary is a great way of visualizing the result of training. The idea is to plot a curve in the input space (which is two dimensional here), such that all points on one side of the curve is classified as one class, and all points on the other side are classified as the other.

`pylab` has a function call `contour` that can be used to plot contour lines of a function given as values on a grid. Decision boundaries are special cases of contour lines; by drawing a contour at the level where the classifier has its threshold we will get the decision boundary.

What we will have to do is to call `mlpfwd` at a large number of points to see what the network would produce at those points. We then draw a contour line at level 0.5 (halfway between the targets 0 and 1 we used during training).

Some matrix wizardry is required to generate a grid of equidistally placed points suitable for `contour`. Here is an example how this can be done. Feel free to copy this more or less literally into your program:

```
1  xrange = numpy.arange(-4, 4, 0.1)
2  yrange = numpy.arange(-4, 4, 0.1)
3  xgrid, ygrid = numpy.meshgrid(xrange, yrange)
4
5  noOfPoints = xgrid.shape[0]*xgrid.shape[1]
6  xcoords = xgrid.reshape((noOfPoints, 1))
7  ycoords = ygrid.reshape((noOfPoints, 1))
8  samples = numpy.concatenate((xcoords, ycoords), axis=1)
9
10 ones = -numpy.ones(xcoords.shape)
11 samples = numpy.concatenate((samples, ones), axis=1)
12
13 indicator = p.mlpfwd(samples)
14 indicator = indicator.reshape(xgrid.shape)
15
16 pylab.contour(xrange, yrange, indicator, (0.5,))
17 pylab.show()
```

Lines 1–2 set up arrays for the sample points in the x and y directions. Line 3 uses `meshgrid` to create two matrices, `xgrid` and `ygrid` which hold the x and y coordinates, respectively, of all grid points.

Now, `mlpfwd` expects all points as rows in one matrix. This is what lines 5–8 are about; `reshape` is used to put all points in a single column, and then the x column and y column are concatenated, side by side. Lines 10–11 append a column where all elements are -1. This is the *bias input* (see section 2.2.2 in the book).

In line 13 we call `mlpfwd` to run the forward pass on all our sample data points. The variable `p` on line 13 refers to an instance of `mlp` which you must first create and train on your data (by calling `mlptrain`; see the examples on pages 56–57 in the text book).

After calling `mlpfwd` we use `reshape` again on line 14 to restore the grid of the result. Finally (lines 16–17), we use `contour` to draw the contour line. Only seeing the contour line not that interesting unless we also see the data points. Use the code from before to plot the data points in the same diagram.

5 Experiments

If you have got all the pieces of the code running, you should now be ready for doing some experiments with the algorithm. There are a number of things you can change:

- The location and number of data clusters in the two classes
- The deviation from the mean of each cluster
- The number of data points used for training
- The number of hidden units
- The step size and number of iterations for training

6 Running and Reporting

This assignment should be reported in a short (two pages are enough) lab report. As a minimum you should include one simple example where you have two well separated clusters, one for each class, and one more challenging classification where a curved decision boundary is required.

For the highest grade you should also present at least one example where overfitting occurs.