Computer Science 152, Fall 2006
Assignment 1

Training Perceptrons

Due Wed. Sept. 6
I will be in Italy the first week, but will try to answer questions via email.

Read the notes provided, and use chapter 3 as a reference. Note that the book tends to use {–1, +1} as the output domain, whereas I have used {0, 1}. The concepts are the same in either case.

As explained in the notes, a perceptron is a single neuron model capable of learning to perform certain functions. Create a self-contained executable program that models the behavior of a perceptron and provides for its training based on input data, using the specification below. You may use any commonly-available language (e.g. C++, Java, SML, …).

Data file description:

The data are free form. Any line orientation present is only to enhance readability. There is no significance attached to an end-of-line.

The first numeral in the file indicates the number of dimensions of the output, say N. For this assignment, this number should always be 1, but in later assignments it might be more than 1 and we want to be able to use the same data sets throughout.

The second numeral in the file indicates the number of dimension of the input, say M. (This does not include a dummy input that you might use against the threshold or bias.)

The remainder of the input is a sequence of *samples*, each consisting of a M+1 numerals. The first numeral is the output of the sample, and will be 1 or 0. The dimension numerals following are the input values. If the last sample has too few numerals, then the program should terminate with the following message:

```
Last sample did not enough values, aborting.
```

In this assignment, the same data is used to both train and test the perceptron. (In the future, there may be a separate file for each.)

Command-Line Description:

The command line has up to three optional parameters:

- A floating-point numeral, the learning rate. If unspecified, this defaults to 1.

- An integer giving a limit on the number of training epochs (complete passes through the training data). If unspecified, this defaults to 100.

- An integer indicating whether to give detailed trace information. If this is positive, the weights and an indication of whether or not they changed will be printed on every training step. If it is 0, only per-epoch information will be printed, giving the number of errors for the entire epoch. If unspecified, this defaults to 0.

Let your input be the standard input, so you don't have to deal with file name parsing.

Don't bother using tags for the command-line parameters; just use bare numeric values.

Your program should run on all of the data sets (with extension "`.in`") that are provided on turing.cs.hmc.edu in `/cs/cs152/data/`. The goal is to get the perceptron to converge in each case where convergence is possible. (Do not use more than 1000 epochs in any case.)

Your output need not match mine exactly, but it should have similar content.

For an input file (`test07.in`) with the following content:

```
1 2
0 1 1
0 0 1
0 1 0
1 0 0
```

meaning that there is one output value, two input values, and the desired function is d(x,y) = 1 if x = y = 0, and d(x, y) = 0 otherwise, the output should something like the following:

```
% percep < test07.in
Input dimension (not counting bias) is: 2
Samples (with desired output first) are:
desired: 1, inputs: 1 0 0
desired: 0, inputs: 1 1 0
desired: 0, inputs: 1 0 1
desired: 0, inputs: 1 1 1
Initial weights:
0 0 0

errors in epoch 1: 2

errors in epoch 2: 2

errors in epoch 3: 1

errors in epoch 4: 0

Final weights:
```

```
0.5 -0.5 -0.5
Weights converged in 4 epochs.
```

If the training fails to converge within the limit of epochs, then the program should end with a message of the form:

```
Weights did not converge in 100 epochs.
```

`test15.in` is a subset of breast cancer laboratory diagnostic data. `test16.in` is the full set. I don't expect the full set to converge.

You are advised to keep your program design clean and modular, because we may subsequently construct programs that build on this basic structure.

If you have problems, the directory also contains an executable `percep` for my version of the program that you can use for comparison.

Turn in on paper an English-language interpretation of the results for the test files, as well as an overall tabular summary and conclusion. Do not simply turn in a print out of your program running; I am interested in a small sample of the detailed running, but not in all details.