Structuring your code for empirical AI research

No one can tell another what is the best way to write a program. Ultimately, everybody has their own preferences and should do it the way that makes sense to them.

Nevertheless, when you start programming for a new purpose, it makes sense to start by programming as others have found useful particularly for that purpose. I have written exploratory programs for AI research many, many times, and have settled on a way of structuring the program that works well for me. I suggest you start by doing as I have learned to do, then modify and enhance that based on your own experience. This might save you some time and bother and, even if it does not, it will make it much easier for me to understand and mark your project.

You have already done the first step, which is to use a high-level, interactive language. The interactive part is crucial, so be sure you have ipython or something similar working before you go any further. The main thing is that you want to be able to easily try things and see what happens. For example, you want to run with one value of alpha and look at the weights, then try another. You want to be able to do things like run with one target function, then, without resetting anything, see what happens if you switch to another. In addition, the interactive environment can just make code development proceed much more quickly because when you are trying to use some unfamiliar function you can type in test cases and quickly see if you are understanding how the function actually works.

Second, I make almost everything global variables, accessible from everywhere. This would include your variables for alpha, x, w, y, z, and n (as given in the project description). Moreover, I suggest you use exactly these mostly-one-character names for these central quantities in your experiment. Do not index them by time (actually, some people do that, and claim it works well; maybe it does, but I have not tried it; for now it is better to stay with tried and true). Thus, alpha, y, and z will be scalars, n will be an integer, and x and w will be vectors (1-dimensional arrays). You should try using numpy arrays so that the learning update can be written very simply and clearly in vector form without any explicit loops.

The big upside to making all the main variables global is that there will be no barriers to you seeing what is going on in your program. This works really well in conjunction with the interactive environment. You will be able to, say, run one training example and then print out x and w to see if your learning update is working properly, or if you understand how it is working. Do this at the beginning with a small n, perhaps 3 or 4, before trying n=20.

Third, I find that it is useful to have functions (as described below) for setting up a series of runs, for initializing a single run, for performing a certain number of the learning steps, for doing a series of runs, and for performing a full experiment. I name these in a stereotyped way. For this experiment, I would use something like the following functions:

`setup(nn);`	Makes the arrays and other static data structures. The first thing it does is set the global variable n to be equal to nn, which is never used again.
`init();`	Initializes the data structures. Called at the beginning of each run. Maybe this is just setting w=0 now, which is trivial and hardly needs to be a function, but later it may get more involved.
`examples(numExamples);`	Presents and learns some number of examples. Returns perhaps a list of the squared error on the examples.
`runs(numRuns,ExamplesPerRun);`	Performs some number of independent runs, each with some number of examples. Returns a list of lists obtained from calls to `examples`, or perhaps their average across runs.
`exper();`	Performs one experiment consisting of a series of calls to `runs` with different settings for the global variables. For example, it might loop over different values of alpha, setting the global alpha to each value, then calling `runs` with that value, collecting all the results for subsequent analysis and graphing.

The last three are generally written in the order given. First you try running examples until you are confident that is working properly and you understand it, then you try multiple runs and parameter values manually. Finally you fire up a longer running experiment with a systematic range of parameter values or other variations.

It also seems to me that for this particular project you might want to have functions like z=target(x) and y=guess(x). Also maybe multiple functions for loading the feature vector in various random ways, perhaps loadXwithBits(x,p), which sets each of the n components of the vector x to 0 or 1—setting to 1 with probability p.

Data collection

You are going to be doing a number of experiments—calling runs(...) a number of times—with different values for global parameters such as alpha and n, and perhaps with different ways of generating feature vectors and targets. How are you to keep track of all of them and have the data collected for examining later and for plotting in different ways?

This is an important topic and deserves some effort to get it right. Later we will develop some software support for it. For now, focus on using your research notebook. Write down what you are doing and what you are trying to do. Include the date and time. If you get some result of interest back from runs or exper, put it in a variable and write the variable name in your notebook; perhaps write the data to a file if you will want it later. Whenever you start to make a significant change in your program, make a copy of your source file (all your code should be in

one file), write it out under a different name, and write the file name in your notebook. Then you will have a record that might include details that you forgot to write down at the time.

Collaboration

Finally, let us work together to be the best we can. If you struggle with something and finally figure it out, let the rest of us know to ease our struggle. If you stumble upon something that might ease the coding, please share that as well. For example, what is the right way in python to get a normally distributed random number? What is a good way to get a graph from matplotlib and then add labels to it manually? We are all learning.