CS 7840 - Soft Computing, Summer 2015
Assignment Four: PCA, Eigenfilters, and Self-Organizing Maps

Objectives and Goals

This assignment is designed to help you become familiar with issues associated with self-organizing maps. These issues include, but are not limited to, PCA and Kohonen Networks. You may complete the work in any programming language you choose.

Deliverables

1. Please prepare and turn in answers to each of the provided discussion questions. You must turn in your materials as a SINGLE zip or tarball via the Pilot on-line learning system. Your answers and supporting materials for this homework assignment must be turned in by 11:59 PM on 17 July 2014. Your archive, referred to later in this document as the "turn-in archive" should be named <your_last_name>_HW4.??? Where ??? is the appropriate extension for the archive type. A zip file that I would turn in, for example, would be named: Gallagher_HW2.zip

2. Your written answers to all questions, including supporting graphics and data tables, should be included in your turn-in archive AS DEFINED in each question/exercise writeup. Please follow directions exactly.

3. If you write any code to help you answer these questions, please submit that code as well inside of your archive. You should refer to the code as needed in your discussion question answers. For example, if you wrote a program called foo.c that helped you answer question 1, in your write up for question 1, tell us the name of the program, where we can find it in your turn in materials, and what it does. You need not turn in copies of programs you did not write.

Homework Questions/Exercises

1. Write code that implements a Hebbian-based Eigenfilter using the *Generalized Hebbian Algorithm (GHA)* found in Chapter 8.5. Your filter should be able to extract the first *four* principal components of a six-component source vector. You may modify the sample code provided or you may start from scratch. Once you have finished your code, construct a synthetic data set that can be used to verify the operation of your code. Explain how you designed your data set, what principle components should be produced via a correct eigenvector extraction from that data, and how you designed that data set so those components would be extracted. For your GHA code, other than my sample code, you *may not* use pre-canned neural networks libraries (E.G. Matlab neural networks toolbox) to help you extract components. You **may**, however, use canned and/or pre-written general-purpose matrix libraries to extract eigenvalues and eigenvectors or to do PCA as part of your plan to **verify** the operation of your GHA linear Eignenfilter network. Turn in your code for the GHA/Linear Eigenfilter AND a PDF file called "question_one_test_plan.pdf" that explains your test data, what eigenvectors should be extracted from it, and output from your program showing those results being produced. Diagrams and graphs would be useful.

2. Let's go back in time a bit and use actual code written by the inventor of the Self-Organizing Map. In the homework #3 materials, you'll find a zip archive for the som_pak package from 1995. I have slightly updated it to be more compatible with modern C compliers and it should compile as is in Linux, MacOS X, and with a little work, under Windows. The rest of the exercises in this homework will be with respect to files you'll find in that "updated" distribution. You can complete those exercises either by compiling and using code in that archive or writing your own from scratch.

   a) Required: In the archive, you'll find a file called `som_pak.pdf`. This is the documentation for the package. Read it.

b) Optional, but recommended: Get a copy of gnuplot and install it on a machine to which you have access (http://www.gnuplot.info/). Gnuplot is a very powerful, open source, plotting program. I use it in some of the shell scripts I added to the package. You may use any plotting package with which you are comfortable, but as mentioned, my scripts use gnuplot, so if you go with that you can use my scripts with no or minor modification.

c) In the DATA subdirectory there are two shell scripts. One of them (`CREATE_JCG_ANIMALS_FEATURE_MAP.sh`) creates a feature map for the file `animals_from_jcg`, which is a modified version of the table 9.2 in the book. Examine the shell script to see the som_pak commands I used. Look into the manual to see what the commands mean and do. Read the manual to come to an understanding of the format of the *.dat files and the *.cod files. I already created a labeled feature map from the trained SOM in `animals_from_jcg.cod.` You may want to back that one up before overwriting it with a new run of CREATE_JCG_ANIMALS_FEATURE_MAP. There might be questions about it specifically.

d) In the DATA subdirectory there is a script called `CREATE_MESH_FROM_FOO.sh`. This shell script creates a lattice visualization of a map trained on the file foo.dat. Examine that shell script and the files it reads and writes. Examine foo.dat data. Examine the mesh. Does the mesh make sense to you?

e) Ok, now a REAL question. Using either som_pak or any SOM training package you like (commercial or self-written), train a self-organizing map on the `animals_from_jcg` data. If you did not use som_pak, output the architecture of the trained map and its weights into the same format used by som_pak *.cod files. The easiest way to do this is to just use som_pak, but you may use other packages or your own code if you wish. Turn in the som_pak format description of your network INSIDE your turn-in archive in a file called `<your_last_name>_animal_map.cod.` This file should not have labels on the neuron weight vectors. See the som_pak documentation.

f) Som_pak has a command called vcal that will label the neuron weight vector in a *.cod file with the identity of an input pattern that it is the champion for (E.G. the neuron that responds most strongly to that pattern of all the neurons in the network). In other words, it generates a text file version of the information in figure 9.10 in the book. You should write a program called v_semantic that creates a labeled *.cod file that is the equivalent of figure 9.11 in the book. In other words, your program v_semantic should write a *.cod file in which EVERY neuron weight vector in the map has a label, and that label is the pattern that the neuron responds most strongly to. You may write this program in any language you like. Note that the C source for the feature_map_gnuplot.c and lattice_gnuplot.c might give you some hints on how to parse and write dat and cod files. Turn in your program AND a labeled cod file for the map you produced in step (e). The labeled cod file should be called `<your_last_name>_animal_map_labeled.cod`

Appendix A: Table 9.2 and Figure 9.10 from 3rd Edition

TABLE 9.2 Animal Names and Their Attributes

Animal		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

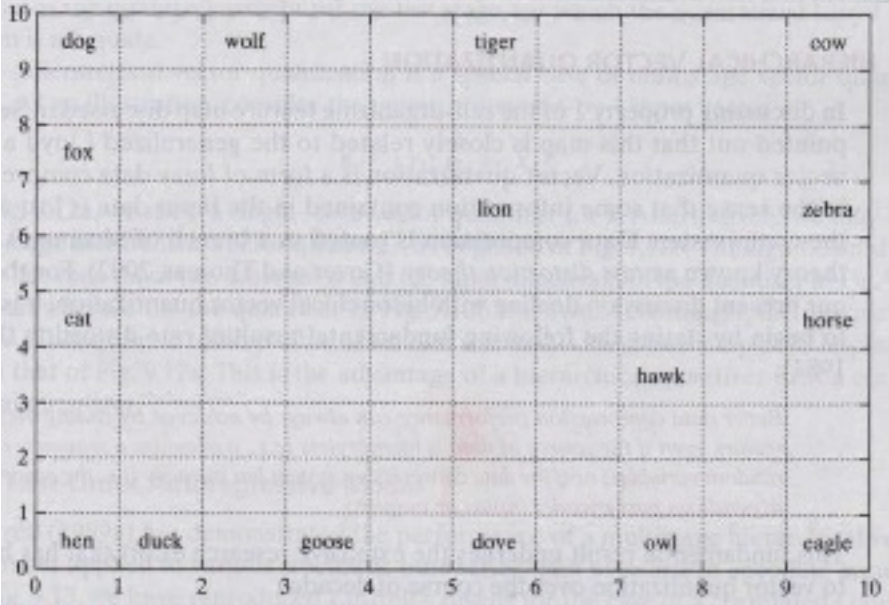

FIGURE 9.10 Feature map containing labeled neurons with strongest responses to their respective inputs.