Homework 1 (26 points total)
Psych 186B, Winter 2018

Assignment Given: Wednesday, Jan. 10, 2018
Assignment Due: 1159pm, Saturday, Jan. 20, 2018

A critical aspect of neural net models in general is the structure and generation of the state vectors used to represent information in the system. We will spend a good deal of time on this issue.

We make the assumption that elements in a state vector are related to the activities of neurons in a group of neurons. In the nervous system, cortical neurons represent highly processed information. The activities of single neurons represent stimulus information that is the result of evolutionary experience and in some cases, the result of some environmental modification during development.

Single unit neurophysiology has given us some insight into what these codings contain -- in the visual system, for example, one finds cells that are motion sensitive, orientation sensitive, etc. What this means for the modeler is that we can legitimately assume that many important aspects of the world have been directly coded for us in the elements of our inputs and we need not worry about deriving them. That is, the representation of information, if done properly, may do much of the work for us. For example, many cells at the low levels of the visual system respond to motion. We can assume this is part of the higher level input vector and we do not need to form the difference between successive images to infer motion directly from image displacement. Presumably the coding used has been evolved to make it easy for higher levels to function. How sensitivity to motion is generated in the nervous system in the first place is an interesting and important question in itself.

At present we know only enough about coding to make very crude descriptions of the low level analysis of the visual system. Other representations used by the brain, especially those involving cognitive material, are almost a complete mystery.

The practical implication of this to a network designer is that we cannot let a smart learning algorithm do our thinking for us. As in any program, you have to have a good idea of what you want the network to do and roughly how you want it to do it. You cannot simply put the data in the input, present the right answer in the output, and expect the system to find the right answers in the future.

In this assignment we will study a simple but useful kind of state vector: one whose elements are random variables. This statistically random vector is easy to analyze and generate and gives useful information about the behavior of distributed systems at a basic level. It also can represent our ignorance of details of interesting codings. Real codings that have statistics like vectors with random elements turn out to be optimal in some situations. In any case, such vectors serve to demonstrate useful properties of distributed systems.

It is easy to generate random numbers on a computer. There is no need to worry about constructing your own generator since there is an adequate one (adequate for what?) in the system library of most computers.

Delphi, for example, has a predefined FUNCTION called **Random** (generates the random value, real or integer) and two PROCEDURES for initializing the sequence, **RandSeed** (sets the initial seed) and **Randomize** (chooses a random seed). The Delphi PROCEUDRE **Randomize** and some other PC random number generators need not use a seed you provide but use the system clock to generate a seed. After the first call, the FUNCTION generates its own future seeds. Too frequent use of **randomize** can produce some exotic errors so beware. The initial seed determines the entire sequence of random numbers. You have to figure out how to provide it conveniently in your program. Remember, sometimes it is nice to be able to reproduce a sequence. All programming languages and computers provide similar facilities somewhere. Be very careful about writing your own random number generator! They are much harder to write than you might think, and some of the ones in common circulation are dangerously flawed.

********** **First Problem** (3 points) **********

1. Write a program to generate a large number of random numbers using your particular random function. As the simplest possible check of the generator, make sure that the values indeed are uniformly distributed between zero and one. Make a histogram of values for say 50,000 calls of the random generator. A simple histogram program with 10 bins can be constructed, truncating, and adding one to the appropriate bin. That is, all values between 0.0000 and 0.0999 will go in the 0'th bin, between 0.1000 and 0.1999 in the first bin, etc. With a uniform distribution, all the bins should contain roughly the same count. Those with a statistical background can analyze this matter further, but this is not necessary to do the assignment. A classic reference to random number generators is found in Donald Knuth's *The Art of Computer Programming, Volume 2.*, Addison-Wesley (2nd. edition, 1981), pp. 1-178, where generators and tests of generators are covered in more detail than you can possibly imagine.

(a) Tell us which function you have used
(b) Generate at least 50,000 random numbers
(c) Plot the histogram

********** **Second Problem** (5 points) **********

2. Uniform random number distributions are not terribly interesting in many applications, but they turn out to be about the only kind you can generate easily by numerical methods. Of course, as everyone is aware, it is impossible to actually generate random numbers on the computer, merely sequences that are sufficiently random for our purposes.

A more generally applicable distribution is the Normal (`Gaussian', `bell-shaped') distribution. It is relatively easy to generate a normally distributed random variable from a uniformly distributed one by a number of means. The simplest way is to add up 12 uniformly distributed random variables, wave your hands, invoke the Central Limit Theorem and claim that a normal distribution is shown by the sum. (An early version of the IBM scientific subroutine package used this approach to generating normal distributions.
Write a program that will return normally distributed random variables. Generate a histogram of values returned by the function. (Remember! These values are no longer necessarily between 0 and 1. You will need more than 10 bins on your histogram and you may have some extreme values.) Compare the values with what you expect from a normal distribution. The exact form, tables

of values, and copious commentary about the normal distribution is found in any elementary statistics textbook.

********** **Third Problem** (15 points) **********

3. Now we have our generators working, we must construct random stimuli. For our purposes, this means constructing vectors with random values. It is usually convenient to construct stimuli with no 'DC' component, that is, the distribution should have mean zero. This means that we should subtract the mean of the vector elements from the element values, or, very slightly differently, doing this probabilistically by subtracting 0.5 from the values generated by Random or your own computer's function giving uniformly distributed random values. In practice, for reasonably large dimensionality, these techniques do not differ in their effects on the neural models.

Your specific assignment will now be to study the behavior of vectors containing random elements. I would like you to generate normalized (i.e. having length 1) random vectors whose elements are taken from a distribution with mean zero. It is probably easiest to use the uniform distribution to generate the elements.

Generate many pairs of such vectors and generate their dot [inner] product.

(a) (1 point) What does this dot product actually mean, geometrically? (Remember, the length of the two vectors is 1.)

(b) (4 points) Generate a histogram of dot products and compute mean and standard deviation of the dot product. Use several dimensionalities: 10, 20, 50, 100 250, 500, 1000 and 2000.)

It is trivial to compute what the mean of the resulting distribution of dot products should be, given the constraints on the vectors.

(c) (2 points) Tell me what it should be (and why) and compare it with your results.

(d) (8 points) Computing the expected standard deviation of the distribution of dot products between these vectors is not so easy but not hard if you know statistics. If you can figure it out mathematically, tell me and compare it with your simulation. (5 points for 2-dimensional and 3-dimensional cases, 3 points for all dimensions beyond. Hint: Start from the definition of variance: $Var(X) = E[X^2] - E[X]^2$. Proceed by applying properties of expected values and what you know about how the vectors were generated.)

(e) (4 points) Otherwise, see if you can figure out (i.e. guess) roughly what it should be from your 'data'. Try to guess how the "width" (standard deviation) of the distribution changes with the dimensionality of the vectors used in the dot product. We would expect the width to decrease as the number of elements increases (wouldn't we?). Some of this material is covered in the textbook.

Reminder: please choose either (d) or (e), but not both.

********** **Fourth Problem** (3 points) **********

4. Please try to estimate a crude value for pi (3.1415926535 ...) using a Monte Carlo method as follows.

(a) Assume a square with side 2 units long centered on zero, that is, the sides run from -1 to +1 along both axes. The area of this square is 4.
(b) Compute two random number (x,y) uniformly distributed between -1 and +1. These two numbers (x,y) correspond to a point in the square.
(c) Assume a circle with radius one, centered on zero, is drawn within the square. This circle has area pi.
(d) Decide if (x,y) is inside or outside the circle.
(e) Do this many many times. Keep track of how many points fall within the circle.
(f) If the (x,y) are chosen from a truly uniform distribution the ratio of total points to the points lying in the circle is the ratio of their areas, that is, 4/pi. This ratio therefore gives an estimate of pi.
(g) Try it out and describe the results.