

Problem Set 4, CNS 185 1999-2000

Handed out: 21 Oct 99
Due: 28 Oct 98

4.1 Associative Memory HKP 11-25, 53-56.

For this problem you will experiment with a 100 neuron associative memory network. You should simulate this network using the `simhop.m` function. The weight matrix will be computed to explicitly store some patterns into the network so that these patterns become the stable states (at least we hope).

The patterns you will try to store into the network are (you guessed it) 10 by 10 images of digits. Each of the ten patterns is a vector of 100 values either +1 or -1. When arranged in a 10 by 10 grid these vectors make binary pictures of the digits 0 through 9. You can get these ten patterns by loading the MATLAB data file `pat.mat`. This loads a matrix `pat` into memory which has 100 rows and 10 columns; each column is a different pattern. We have provided a function to display these patterns; use `disppat(pat(:,k))`.

1. Write a routine which computes the weight matrix T according to the outer product formula. Namely, if \vec{P}^k are the memories (patterns) to be stored, first let

$$T = \sum_{k=1}^m (\vec{P}^k)(\vec{P}^k)^T, \quad (1)$$

then set all diagonal elements to zero ($T_{ii} = 0$ for all i). Here m is the number of patterns to be stored, in our case $m \leq 10$. Give a brief answer to the question: “Why are we setting all the diagonal elements to zero?” (note: it is not required for stability). Later you will need to experiment with storing different numbers of patterns, so be sure the variable m is easy to change in your routine.

For these simulations, you will set initial conditions for the \vec{u} variable and read output as the \vec{u} and \vec{V} variables. Start the differential equations in their initial state and simulate the network until convergence. For all practical purposes, you can guarantee convergence after a reasonable number of iterations (say 100). We have provided a function `plothopd.m` which shows you a movie of the network converging so you can get an idea of what is happening. (In general, it is possible to follow the convergence at each time step by using the Lyapunov function, but it takes a little manipulation to get the second term of this function into a transparent form¹. If you do this you should see the Lyapunov function always decreasing.)

¹Notice that $\int_0^V \tanh^{-1}(z) dz = V \tanh^{-1}(V) + 1/2 \ln(1 - V^2)$ and recall that $\tanh^{-1}(x) = 1/2 \ln[(1 + x)/(1 - x)]$.

2. Now we would like you to simulate the network with a weight matrix T_{ij} constructed from the first m of the 10 total memory patterns provided; observe the behavior when you store $m = \{1, 2, 3, 5, 7, 10\}$ patterns. For each value of m above, try a few different *random* initial states and simulate the network until it converges. Generate the random initial states as vectors where each element is drawn independently from a zero mean, unit variance Gaussian (in MATLAB use `randn`). The network should ideally converge to one of the stored patterns but sometimes spurious states crop up as attractors. To check this out, display and hand in the final stable states of \vec{V} as pictures.
3. Briefly describe your findings. For what value of m does the network start to fail? How does this value compare with the theoretical capacity of a 100 neuron network (see page 19 in HKP)? What do the spurious states look like?
4. For the largest value of m for which the network correctly stored all m memories, try the following: Start the network in a corrupted version of one of the stored patterns. In other words, create an initial condition vector by adding a stored memory vector to a random state like the ones you have been experimenting with. In these cases, the network should ideally “restore” the corrupted pattern; in other words it should behave as an associative memory. Experiment with different levels of signal-to-noise by adding a constant times the original pattern (*i.e.*, a fainter or stronger signal) to a zero mean unit variance Gaussian random vector. For one particular digit, determine how “faint” the initial pattern can be before it fails to be correctly retrieved. Express “faintness” as the ratio of noise variance to signal variance. (Be careful when you calculate the signal variance not to instead compute the signal standard deviation.)

The reason the associative memory does not work very well is that the patterns we are trying to store are highly *correlated*. (Think about why this might be; see HKP pp. 17-20 for more details.) What does this mean? It means that they often have the same pixels turned on or off. For example, the leftmost and rightmost columns of the image are always +1 for every pattern. You can see this more clearly by computing the matrix of pattern inner products: this is a 10 by 10 symmetric matrix whose ij^{th} element is the dot product between memory i and memory j . If the patterns were perfectly decorrelated, this would be 100 on the diagonal and zero everywhere else. But in fact, there is significant correlation.

5. Compute this matrix Z of inner products. Hand in a picture of it produced with `colormap('gray'); imagesc(Z, [0 100])` which shows the correlations between patterns. (Hint: you can compute Z in one line with a simple matrix operation on `pat`. Don't use for loops.)

It turns out that we can *rotate* the axes of the original patterns in order to decorrelate them. The *optimal* rotation to do this, as you will soon learn, is by using PCA! Unfortunately, 10 patterns is not enough data to estimate the required 100 by 100 rotation matrix using PCA. So your friendly TAs have provided a special function called `magicrot.m` which computes a decorrelating rotation for you. Below you will use it to substantially increase the capacity of the Hopfield net associative memory.

6. Compute the decorrelating rotation matrix R by calling `R=magicrot(pat);`. Now make a new set of “uncorrelated patterns” by rotating the originals: `puncorr = R*pat;`. Look at a few of these using `disppat;` they should look much more random. Again, compute the matrix Z of inner products for this new set of patterns and hand in a picture of it produced with `colormap('gray'); imagesc(Z,[0 100])`. What do you notice about this new Z matrix? (Your TAs did a good job, huh?)
7. Rerun your associative memory by storing and retrieving patterns as follows: Compute a new T_{ij} which stores the uncorrelated patterns `puncorr`. Now modify `simhop` so that when you want to initialize the network to a particular initial state you first “rotate” this initial state and then present it to the network. Run the network dynamics (using the *new* T_{ij}) until they reach a stable point and then “unrotate” the resulting stable state by multiplying by the inverse rotation matrix `inv(R)`. How many of the ten patterns can you successfully store and retrieve in the network this way? With this maximum number of patterns stored, in the presence of noise how faint (again in the signal-to-noise variance sense) can initial patterns be and still be correctly retrieved most of the time? (Big improvement, huh?)