3.1 Associative Memory

For this problem you will experiment with a 100 neuron associative memory network. You should simulate this network exactly as you did in the previous problem for the 2 neuron case; use the same simhop.m function. In this case, however, 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 dispMAT(pat(:,k)) (careful not to use the old disprow because it has some odd autoscaling built into it).

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

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

then set all diagonal elements to zero ($T_{ii} = 0$ for all $i$). \(^1\) Here $m$ is the number of patterns to be stored, in our case $m \leq 10$. 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.

- As with the previous simulations, you will set initial conditions for the $u$ variable and read output as the $u$ and $V$ variables. Start the differential equations in their initial state and simulate the network until convergence. Notice two things: we have not specified a value of the gain $\beta$ for you to use – merely choose one that makes things relatively high gain. Also, we have not told you whether to interpret the $\pm 1$ values as $u$ or $V$ values – it turns out to not matter much so just choose one and state your assumption. For all practical purposes, you can guarantee convergence after a reasonable number of iterations (say 100). We have provided a function plotHop.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 form\(^2\). If you do this you should see the Lyapunov function always decreasing.)

- 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 (but do not hand in) the final stable states of $V$ as pictures.

\(^1\)For extra credit, give a brief answer to the question: “Why are we setting all the diagonal elements to zero?” (note: it is not required for stability)

\(^2\)Notice that $\int_0^V \text{arctanh}(z)dz = V \text{arctanh}(V)+1/2\ln(1-V^2)$ and recall that \text{arctanh}(x) = 1/2\ln[(1+x)/(1-x)].
• 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?\(^3\) What do the spurious states look like?

• For the largest value of \( m \) for which the network “correctly stored” all \( m \) memories, try the following: Start the network in a 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 (ie 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.)

3.2 Optimization by Networks: Graph Bipartitioning

Consider the problem of partitioning a graph with \( N \) vertices (\( N \) even) into two groups, such that the number of edges between the two groups is minimized, subject to the constraint that the number of vertices in each group is equal. We can write an energy (cost) function which can then be locally minimized by the dynamics of our neural network, just as in the case of the associative memory last week. A good representation is for neuron \( i \) to be on \((V_i = +1)\) when vertex \( i \) is in group one, and off \((V_i = -1)\) when vertex \( i \) is in group two. A good representation of an arbitrary graph is to let

\[
C_{ij} = \begin{cases} 
1 & \text{if there is an edge between vertex } i \text{ and vertex } j \\
0 & \text{otherwise} 
\end{cases}
\]

Recall that in the high gain limit our Lyapunov function is

\[
L = -\frac{1}{2} \left( \sum_i \sum_j T_{ij} V_i V_j \right) - \left( \sum_i I_i V_i \right)
\]

and that \( L \) is always reduced as the network evolves according to its dynamics.

• Show how to construct \( T_{ij} \) and/or \( I_i \) in terms of \( C_{ij} \), so that a global minimum of \( L \) gives a solution to the graph bipartitioning problem. Your formula must be \textit{explicit}, with no free parameters (if you have a free parameter, you must specify for what range of parameters the formula will work).

![Graph Example]

• Simulate a differential equation that solves the problem. Try it on the graph above. (Hint: you can reuse some neural net software you have previously written. Also, there is a choice of constants that makes connections especially simple). What is the partitioned result? What is the gain \( \alpha \) you used in the function \( V_i = g(u_i) = \tanh(\alpha u_i) \)?

\(^3\)The theoretical capacity is \( \frac{N}{4mN} \), in which \( N \) is the number of neurons.