CNS 187 - Neural Computation
Problem Sheet 4

Handed out: 24 Oct 00
Due: 30 Oct 00, 5pm

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 main line for Euler integration has been left blank; you must fill in this line. 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 W according to the outer product formula. Namely, if $\vec{P}^k$ are the memories (patterns) to be stored, first let

$$W = \frac{1}{N} \sum_{k=1}^{M} (\vec{P}^k)(\vec{P}^k)^T, \tag{1}$$

   then set all diagonal elements to zero ($W_{ii} = 0$ 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 $\vec{V}$ variable and read output as the $\vec{V}$ and $\vec{S}$ variables. Start the differential equations in their initial state and simulate the network until convergence. What gain should you use? What `eta` should you use? For all practical purposes, you can guarantee convergence after a reasonable number of iterations (100 is probably overkill). 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 form[1]. If you do this you should see the Lyapunov function always decreasing.)

[1] Notice that $\int_0^S tanh^{-1}(z)\, dz = S\, tanh^{-1}(S) + 1/2 \ln(1 - S^2)$ 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 W^M constructed from the first M of the 10 total memory patterns provided; and you will observe the behavior as a function of M. You will do this by examining the attractor-basin structure of the phase space. Let $f(\vec{S})$ be the stable fixed point that the network converges to when started in state $\vec{S}$. We say that $f(\vec{S})$ is a *correct memory* if $\pm f(\vec{S}) = \vec{P}^k$ for exactly one k. We say that $f(\vec{S})$ is a *confused memory* if $\pm f(\vec{S}) = f(\vec{P}^k) \neq \vec{P}^k$ for exactly one k. Otherwise, we say that $f(\vec{S})$ is a *spurious memory*. We are interested in what fraction of the volume of $\vec{S}$ space leads to correct, confused, and spurious memories. You will want to write a MATLAB routine to automatically classify fixed point memories.

   By simulating *random* initial states chosen with $S_i \in (-1, 1)$, estimate the fraction of the volume of $\vec{S}$ space that leads to correct, confused, and spurious memories. (Note: be sure to transform your random $\vec{S}$ into $\vec{V}$ space for the simulation.) Plot these fractions as a function of M, including error bars indicating the standard deviation of your sample. Also plot the number of desired patterns that are correct memories.

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 pages 19 and 39 in HKP)? What do the spurious states look like?

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 pattern i and pattern 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.

4. 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.) Recall that a pattern P^k is stable iff

$$P_i^k = sgn\left(\sum_j w_{ij} P_j^k\right) = sgn\left(P_i^k(1 - C_i^k)\right).$$

   Explain how the matrix Z is related to the cross-talk terms C_i^k.

In class, we estimated the capacity of an associative network by considering randomly-chosen patterns, and then estimating the probability that a chosen pattern is stable. Key to our argument was the fact that random patterns are only very weakly correlated. We would like to transform our desired memory patterns – the digits – into weakly correlated patterns, using a simple linear one-layer feedforward network. That is, pattern $\vec{P}^{in}$ is transformed into $\vec{S}(0) = T\vec{P}^{in}$, i.e., $S_i(0) = \sum_j T_{ij} P_j^{in}$. The recurrent network dynamics then bring $\vec{S}$ to a fixed point attractor. Another linear one-layer network transforms the fixed point $\vec{S}(\infty)$ back to the original coordinate system: $\vec{P}^{out} = \hat{T}\vec{S}(\infty)$. This can roughly be envisioned as a rotation of the coordinate system in which the patterns are represented.

5. We will construct a linear transformation that does the job. To find a "randomizing" matrix T, choose a matrix of random patterns R and require that $R = TP$, where P is the $N \times M$ matrix of vectors $\vec{P}^k$. (Hint: it may help to "pad" P with extra random patterns, so that P^{-1} can be computed.) What is $\hat{T}$? In MATLAB let `puncorr = T*pat;`. To confirm that you did this right, compute the matrix Z of inner products for this new set of patterns – is the magnitude of the inner products what you would expect?

6. Rerun your associative memory by storing and retrieving patterns as follows: Compute a new W_{ij} which stores the "randomized" 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* W_{ij}) until they reach a stable point and then "unrotate" the resulting stable state using $\hat{T}$. Again, estimate the fraction of the volume of $\vec{P}$ space that leads to correct, confused, and spurious memories, and plot these fractions as a function of M with error bars. Also plot the number of desired patterns that are correct memories.

7. 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 – see how well the memory works 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. Provide these input patterns to the network in $\vec{V}$ space, since we aren't guaranteeing that our patterns are bounded between -1 and +1. 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.)

8. We didn't really "uncorrelate" the memory patterns as much as we could. Show (on paper) how to find T and $\hat{T}$ that *perfectly* decorrelate the patterns memorized by the associative network. You may assume N is even and that a set of M orthogonal ± 1 vectors are known. Calculate the crosstalk terms C_i^k when the memorized patterns are orthogonal (i.e., $PP^T = NI$ where $P_{ik} = P_i^k$), and show that up to $N - 1$ stored memories can be stable. What is W when N memories are stored?

4.2 Winner-Take-All Networks and Nearest-Neighbor Memories

Using extra input to each units, we can create an M-unit network with exactly M stable states: each stable state has one unit "on" ($S_i = +1$) and all the others "off". This can be accomplished by providing inhibitory connections between all units while exciting all units with a tonic input. Specifically, $W_{ij} = -1$ for $i \neq j$, $W_{ij} = 0$ for $i = j$, and $I_i = \alpha$ for some α to be determined, where the dynamics of the system are

$$\dot{V}_i = -V_i + \sum_j W_{ij} S_j + I_i$$

and $S_j = \tanh(\beta V_j)$.

1. Note that by symmetry, the Lyapunov function $\mathcal{L}$ should be a function only of the number of units, m, that are "on". Write down the high-gain Lyapunov function in terms of m. Determine what values of α guarantee that the only stable states have exactly one unit "on". For convenience, let $\vec{S}^k$ be the pattern with only the k^{th} unit "on".

2. For such a value of α, determine the attractor basin boundaries in the continuous dynamics. That is, characterize the set of initial states $\vec{V}$ such that the fixed point is, say, $\vec{S}^1$. Hence the name, "Winner-Take-All" (WTA).

Let us use the WTA network to create an associative memory. Again, consider a simple linear one-layer network mapping the N-bit input patterns $\vec{P}^{in}$ to the M WTA units, and another linear one-layer network mapping the WTA units to the N-bit outputs $\vec{P}^{out}$. In this case, we will modify our dynamics so that

$$\dot{V}_i = -V_i + \sum_{j=1}^{M} W_{ij} S_j + I_i + \sum_{j=1}^{N} T_{ij} P_j^{in}$$

and the output units behave according to

$$\dot{U}_i = -U_i + \sum_{j=1}^{M} \hat{T}_{ij} S_j$$

and

$$P_j^{out} = \tanh(\beta U_j).$$

Again, we want to memorize patterns $\vec{P}^1 \ldots \vec{P}^M$.

3. Write an analytic form for the equilibrium V_i in the low-gain limit.

4. Find T such that for every k, at low gain V_k is the most active unit if $\vec{P}^k$ is presented as input. (Note that the patterns to be memorized all have norm $||\vec{P}^k|| = \sqrt{N}$.)

5. Describe what happens to S_i when the gain is suddenly turned from very low to very high.

6. Find $\hat{T}$ such that at high gain, if the WTA is in state $\vec{S}^k$, then the output is $\vec{P}^k$.

7. Describe the behavior of this network as an associative memory. Hence the name, "nearest neighbor memory". Another name: "grandmother cells". Compare the behavior of this memory to the recurrent associative network you used in the first problem, assuming a set of random patterns. How large can M be, compared to N? How many weights are used per bit memorized? For cases where both memories can store all the memories, what behavior would you expect if a small fraction of the weights in each network were "damaged" by setting them to zero?

4.3 Translation-Invariance by Fourier Transforms

Suppose we would like to recognize a pattern even if it occurrs translated in a 2-dimensional image. One idea would be to pre-process the input patterns to a translation-invariant representation. The two-dimensional Discrete Fourier Transform does the trick.

Suppose the $N \times N$ image is $a(x, y)$. Then

$$F_a(f_x, f_y) = \frac{1}{N} \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} a(x, y) e^{\frac{2\pi}{N} i x f_x} e^{\frac{2\pi}{N} i y f_y}$$

where $0 \leq f_x, f_y \leq N - 1$. Note that F_a is typically complex-valued.

1. Show that if $b(x, y) = a(x + d_x, y + d_y)$ where addition is performed mod N, then $||F_a(f_x, f_y)|| = ||F_b(f_x, f_y)||$.
2. What problems would you expect to encounter if you were to try this trick with the digits?