The Johns Hopkins University

JHU ENGINEERING FOR PROFESSIONALS PROGRAM

NEURAL NETWORKS: 625-438/605.447

Problem Set #10

I. Problem Statement

Develop a Boltzmann machine model and test it using a programming language that can solve the Traveling Salesman Problem (TSP). In your model, use 5 cities with the following distances between cities and use the results of your program to answer the following questions:

	City to City Distance				
	A	B	C	D	E
A	-	10	20	5	18
B		-	15	32	10
C			-	25	16
D				-	35
E					-

a) State the tour with the minimum total distance.

b) State the minimum distance associated` with the optimal tour from part a.

Develop your own code and run you experiments in the way you see fit. Refer to the reading assignment on the best ways to model your Boltzmann machine. This reading assignment is essential and yet a bit obtuse. In this set of instructions, I will fill in some gaps so that you can write your code and do some experimentation. Feel free to compare your results (but not your code) with your classmates. This feedback can help you determine whether your experiments are producing reasonable answers. Note also, that the optimal solution is easy enough to determine "by hand" and you should use this to gauge your progress in your computational experiments.

II. Problem Considerations

To do this problem, you need to take into account a number of important things in order to develop an appropriate mode and corresponding program. First, you need to fully understand what the TSP entails --- that is, how a Hamiltonian Tour (HT) is defined. A solution to the TSP is a Hamiltonian Tour with minimum total distance and is defined as a sequence of cities the salesperson visits such that the total distance traveled is minimized, every city is visited only one time and the salesperson returns to the city he/she started from. From the distance matrix above, an example of an HT would be a sequence of cities such as

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow A$$

There are many ways to define the solution to the TSP as a math program where the objective is some objective function to be minimized subject to various constraints. The most obvious function to use would be the Consensus function from the lectures. Before we get into that however, lets examine ways to model the problem first.

To define the proper decision variables that will correspond to states of nodes in a Boltzmann Machine, consider the following figure to help you understand important points. This figure is inspired from the reading assignment and we will provide a bit more detail here.

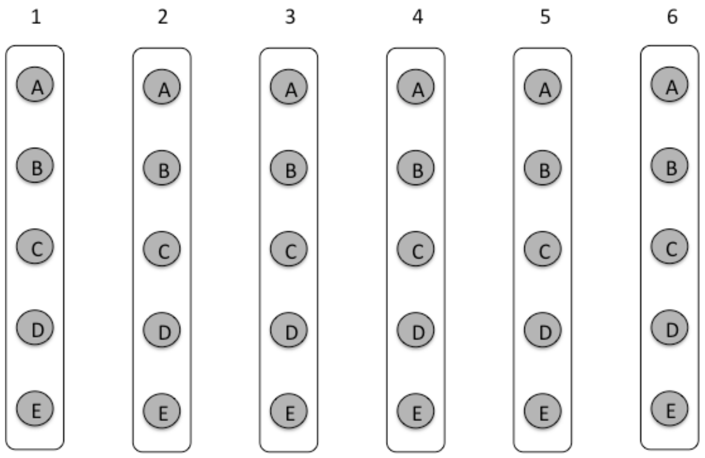

Figure 1.

Figure 1 depicts a set of 30 Perceptrons indicated as the circles in a set of rectangles with the letter corresponding to a city. Each rectangle corresponds to a time epoch indicated with the numeral above the rectangle. An HT therefore corresponds to a sequence of cities such as the one noted in Figure 2. Note that the city indicated in Epoch 1 is the same city as indicated in Epoch 6 --- where the salesperson is returns to homebase, *i.e.,* the point of origin. The cities visited in each epoch are filled in as green nodes. This corresponds to states of the Perceptrons in a Boltzmann Machine being equal to 1.

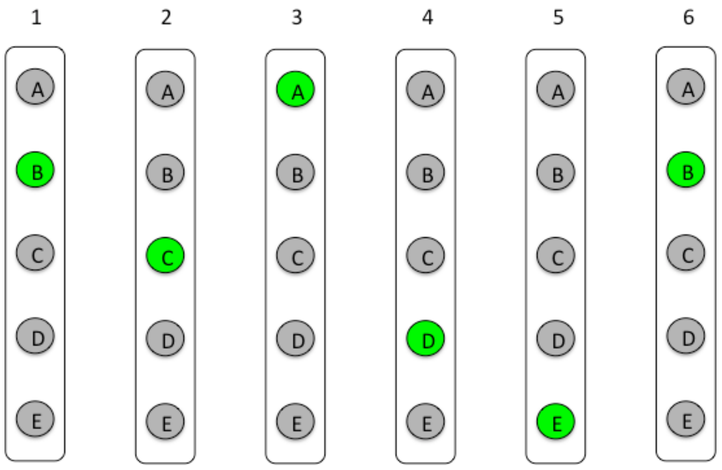

Figure 2.

Figure 2 depicts the HT of B $\rightarrow$ C $\rightarrow$ A $\rightarrow$ D $\rightarrow$ E $\rightarrow$ B.

There are several things to take note of in Figure 2. First, the length of the HT and even the HT itself doesn't require any specific starting point. Any city will do, but the "on" city in Epoch 1 must be the same as the "on" city in Epoch 6. Consequently, the only real decision variables correspond to the node states in epochs 2 – 5. That is, your code will randomly select states for these Perceptrons and calculate the total distance corresponding to each configuration of states.

Another important thing to be aware of is that in any HT, and here corresponding only to epochs 1 – 5, there can only be one Perceptron state being "on" in any row or column. This corresponds to the inherent constraints of the problem that each and every city is visited and only one time. This also implies that no two cities are visited during any given epoch, hence only one Perceptron is on in any epoch (rectangle). Mathematically, this corresponds to a weight matrix that is *unimodular* where the sum of any row or column equals 1.

Since these nodes are part of a Boltzmann Machine, they are all connected and this also means the associated weights are symmetric since the distance from city A to city B is the same as the distance from city B to city A. But how should we go about modeling the TSP with a Boltzmann Machine? How do we set up the weight matrix and calculate activity function values and Consensus function values? Let's first consider how we should define each Perceptron and each Perceptron state. First, each Perceptron corresponds to a city and is associated with a time epoch. Thus, we can define a state variable x with two subscripts thusly:

$$x_{city,epoch} = \begin{cases} 1 & \text{if salesperson visits 'city' at time epoch 'epoch'} \\ 0 & \text{otherwise} \end{cases}$$

So that's our basic state variable in the Boltzmann Machine. Now we have to figure out something about an objective function.

In the Hecht-Nielsen function we must include a set of weights. These weights can be symbolized as $w_{[city,epoch],[city,epoch]}$ which corresponds to a weight from one node to another node. That is, this weight corresponds to the link between one node in one rectangle and another node in some rectangle. How shall we assign its values? Since every node in a Boltzmann Machine is connected to every other node, we have to figure out this assignment values and this is central to how a Boltzmann Machine *anneals* to the best solution. Consider this example: $w_{[A,2],[D,3]} = 5$. This makes sense if you look at Table 1. Moving from city A at epoch 2 to city D at epoch 3 involves a distance of 5 miles from the table. We would somehow add this to a total that is computed as the HT is annealed in the Boltzmann Machine. If the total distances are minimized, then we get the optimal solution. Recall that the Consensus function is defined as

$$C = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j - \sum_{i}\theta_i x_i$$

This can serve as our objective function in a type of simulated annealing program executed on a Boltzmann Machine! If the weights correspond to the inter-city distances, then *minimizing* the Consensus function should produce the shortest HT. Right? Well, there are unfortunately a few issues we have to deal with. First, if we use the Boltzmann Machine the typical way and assign states randomly, anneal, reduce temperature, etc. and do everything we're supposed to do in running simulated annealing, would the minimum distance be computed when all states are zero?! Obviously, the first part of the Consensus function would end up being zero, a pretty small number. But that isn't going to correspond to a set of decision variables *x* that corresponds to a valid HT. Perhaps if we use the bias terms wisely, we could force the situation to anneal naturally so that the minimum Consensus function value only occurs when HT valid states are equal to 1.

To shed light on this idea, we need to figure out how to define weights so that the Consensus function that we are trying to minimize is *increased* by a lot when certain states are "on" that shouldn't be "on". That is, we want to increase the function when states violate the constraints of the TSP. Consider for example the transition weight $w_{[A,2],[D,2]} = ?$ What should we set that equal to? Obviously, there is a link between Perceptrons [A,2] and [D,2] (remember, each Perceptron has a city and time epoch subscript) and we have to assign it some value. This weight corresponds to a link between two nodes in the same epoch! We can't allow our Boltzmann Machine to produce these state values where both $x_{A,2} = x_{D,2} = 1$ as might happen as we randomly assign state values to the Perceptron. If we add a lot to the Consensus function for those terms that violate constraints there is less *likelihood* that the simulated annealing algorithm used in the Boltzmann Machine would visit such a configuration. So if those state variables are both 1 (which we don't want to happen) we should set the weight $w_{[A,2],[D,2]}$ that links these two nodes to be a very high positive value, higher than say the longest distance from the distance table. This will then inhibit such a configuration.

We can also come up with a similar rule for links between two nodes for the same city, but at different time epochs. Thus, $w_{[B,2],[B,5]}$ corresponds to a link between two nodes corresponding to visiting city B at time epochs 2 and 5. We can't have that either since we only visit a city once. Thus, this weight also should be high.

The final issue is to figure out how to minimize the Consensus function without that function being minimized by setting all state variables to zero. Then we would not be visiting any city. Consider what the Consensus function looks like for just two cities A and B. Then

$$C = w_{[A,1],[B,2]} x_{A,1} x_{B,2} - \theta_A x_{A,1} - \theta_B x_{B,2}$$

Here, the weight corresponds to the distance since the transition is a legitimate transition. Now if both state variables are 0, then $C = 0$. Can we get C even smaller when both state variables are 1?

III. Program and Assignment Requirements

Using the information in Parts I and II above, write a program that solves the TSP using a Boltzmann Machine. This will require you to start out with a high temperature and decrease it according to some cooling schedule. For this problem, you will be slowly annealing a solution by minimizing the Consensus function.

The program you write should have plenty of commenting lines. Each iteration of the program should print out a line indicating the total distance travelled and the corresponding sequence of cities. Feel free to use asynchronous updating for each iteration. That is, randomly select a node, calculate the change in Consensus function value if the node state is changed, apply the Metropolis Acceptance Criterion to determine whether to accept the state change. If that change is not accepted, randomly select another node (it may end up being the same one) and so on. If the node state change is accepted, compute the new Consensus function value and print out the corresponding solution, *i.e.,* the distance and sequence of cities. Then, reduce the temperature according to the cooling schedule and repeat the process. You may also want to add code that stores the best solution found as your program runs. The printout should then show a steadily improving solution to the TSP. Good luck.

IV. Grading Considerations

Your grade on this assignment will be based on how easily it is for me to understand your code (commenting helps a lot), the quality of the code you write and the output. Remember, keep things as simple as possible. The 'handed-in' assignment should be comprised of the code and the output produced by your code. You may also include a brief synopsis of how you assigned weights. Good luck.