EECS/EEAP 484 Computational Intelligence, Fall 2015
Problem Set 9: Recurrent Neural Networks for Optimization

In problem-set 8, we saw that neural networks with feedback (recurrent neural nets) can be used for error-tolerant data encoding, content-addressable memory, and recovery of complete memories based on incomplete or flawed cues. In this problem set, recurrent neural nets will be used as a means for solving combinatorial problems.

This assignment follows the presentation by Hopfield and Tank in *'Neural' Computation of Decisions in Optimization Problems*, which is posted on Blackboard. Some example code is on the class website, along with data describing a 10-city distance matrix, "intercity_distances.dat". All 10 of the hypothetical cities lie on a plane within a square from (0.0,0.0) to (1.0,1.0). For these types of problems, people are good at visualizing effective solutions. However, the distance map need not be so geometrically intuitive, but could represent transition costs that do not scale intuitively with distance (e.g., as in airline pricing). In such non-geometric cases, people are not good at visualizing good solutions.

Since the 10-city tour is presumed to be cyclic (return to the start city after visiting all cities), the choice of the start city is irrelevant. For example, a tour that travels among cities in the order A,B,C,D,E,F,G,H,I,J,A will have exactly the same path length as the tour B,C,D,E,F,G,H,I,J,A,B. For the 10-city tour, there are 10 such identical sequences. For a symmetric distance map (cost of travel from A to B is identical to the cost of travel from B to A), there are another 10 tours in the reverse order that are identical in cost.

It is thus sufficiently general to mandate starting from city 0 and ending at city 0, which reduces the search size to 9!=362,880 permutations. This is a relatively small solution set that can be evaluated exhaustively. However, adding just a few more cities makes the problem intractable, and one requires some other optimization technique.

Note that in Hopfield and Tank's paper, equation 12 contains a term: $-U_{xi}/\tau$. You can try using this term, or try eliminating it. Also, you will need to pick values for the weights A, B, C, and D, and for the steepness parameter of the activation function, U0. You will also need to pick a time step for integrating values of U based on dU/dt. Further, note that Hopfield and Tank commented that they used a larger value for the current-source terms than that corresponding to their analytic derivation (page 147).

Watch out for convergence. You might find that your set of outputs settles on values that do not satisfy the properties of a valid path. I will suggest some optional code, using integral-error feedback, that ultimately forces convergence to legal tours. You may experiment with this option if you find it useful.

You should do the following:

- Complete the code: fix the function “assign_weights.m” to correctly include the constraint influences from A and B terms.

Try the code Hopfield_TSP.m, with compute_udot.m including and excluding the line:

```
%U_dot(X,ix) = U_dot(X,ix)+(LAMBDA)*int_Eabc(X,ix);
```

- Describe statistics for performance of your neural-net computation of solutions. Obtain 100 solutions from your network and summarize the statistics of the results. Compare to the exhaustive solution set (which has been provided for you).

- Report on your choices for all parameters and describe how/why you chose these parameters in terms of your network performance. Describe any variations you introduced.