Using Hopfield Neural Network for Solving Travel Salesman Problem

Nikolai Zakharov
Department of Computer Science and Technology
Tsinghua University
Beijing
luoyy16@mails.tsinghua.edu.cn

Abstract

Travel salesman problem is classical optimization problem. In this project we used Hopfield neural network for finding the optimal solution of this problem. This is final project of the "Intelligent control" class.

1 Introduction

A large class of logical problems arising from real world situations can be formulated as optimization problems, and thus qualitatively described as a search for the best solution. These problems can be found in engineering, commerce and many other fields. The "Traveling-Salesman Problem" (TSP) is a classic of difficult optimization. It‘s very simple to formulate but hard to solve. There is a list of cities that are to be visited by a salesperson. He starts from the first city and after visiting other cities must return to the city it started his journey. He also cannot visit the same city twice on his route.The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents traveling times or cost, or a similarity measure between DNA fragments. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. In many applications, additional constraints such as limited resources or time windows may be imposed.

2 Definition of the problem

More formally we have a set of N cities A, B, C... which have distances d_{AB}, d_{AC}.... The problem is to find a closest tour which visits each city once, returns to the starting city, and has a short(minimum) total path length. A tour defines some sequence of visited cities (ex. B, E, D, ...) and the total path length of the tour is defined as $d = d_{BE} + d_{ED}$.... In the theory of computational complexity, the decision version of the TSP belongs to the class of NP-complete problems. Thus, the time required to solve this problem on any computer grows exponentially with the number of cities. Also there is no algorithm which gives a perfect solution so we look for some approximate methods for solution of this problem.

3 Proposed solution

For solving this problem, Hopfield and Tank[1] proposed using Hopfield neural network for solving this task. Hopfield neural network is a recurrent neural network, so connections in the network form cycles. The units (nodes) in a Hopfield network can have only 2 values: $V \in \{0, 1\}$ or $V \in \{-1, 1\}$, but in either case values that encode a binary state. The network is fully connected (each node is connected with any other node). The connection matrix is symmetric, that is the weights on the edges are the same in both directions. Figure 1 shows a Hopfield network for 3 cities travel salesman problem.

The neural network used to encode TS problem and it's solution to a single squared matrix having N

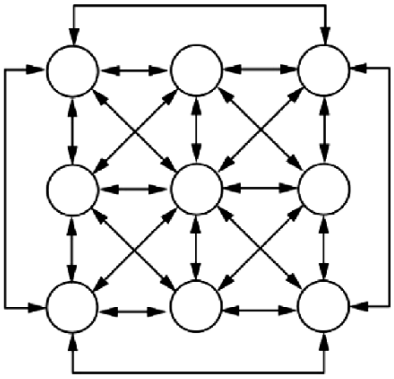

Figure 1: Hopfield NN for 3-cities TSP

rows, which represent cities, and N columns, which represents position in the tour sequence. After neural network converges to it's final state each value will be equal to 0 or 1. Initial Hopfield network formulation had only binary values for neurons(0 or 1), but for solving TSP [1] used continuous values on range $V \in [0, 1]$. The value of each node V depends on the input potential of the node U. For keeping values between desired range we used sigmoid function (fig.2):

$$V_{Xi} = g(u_{Xi}) = \frac{1}{1 + \exp(-2\frac{u_{Xi}}{u_0})} = \frac{1}{2}(1 + \tanh \frac{u_{Xi}}{u_0})$$

where u_{Xi} is the net input of neuron (X(city), i) and u_0 is a parameter determining the steepness of the gain.

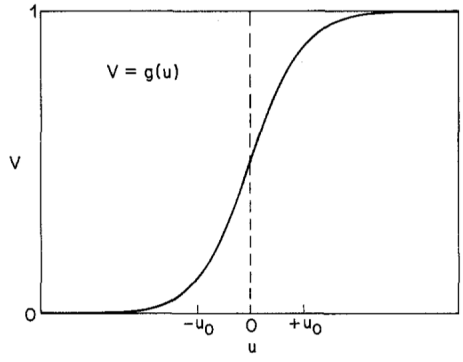

Figure 2: The input-output relations for neurons

The value of the input potential(u) of each node depends on the values of all nodes that are connected to it and the values of the connections weights. The initial neuron input potential was set to:

$$u_{int} = -\frac{u_0}{2} ln(n - 1)$$

Over algorithm iterations, each node(neuron) value will change. Algorithm will converge when the matrix of nodes will satisfy following criteria:

- Only one node in each row will have value 1
- Only one node in each column will have value 1
- There will be exactly N nodes having value 1
- Distances between cities in a trip will be minimal

Each state of the network can be described as a single value called Energy. The energy function must take into account above rules[2]. So [1] proposed the energy function in the following form:

$$E = A\sum_{X}\sum_{i}\sum_{j\neq i} V_{Xi}V_{Xj} + B\sum_{i}\sum_{X}\sum_{Y\neq X} V_{Xi}V_{Yi} + C(\sum_{X}\sum_{i}\sum_{Xi} V_{Xi} - N)^2$$

$$+D\sum_{X}\sum_{Y\neq X}\sum_{i} d_{XY}V_{Xi}(V_{Y,i+1} + V_{Y,i-1})$$

Indices X and Y refer to cities, and i and j refer to days, and the sums run from 1 to N. The first three terms enforce the syntax of the solution(first 3 criteria from above) and A,B,C,D are network parameters.
Input potential of nodes can be described by:

$$u_i = -\frac{\partial E}{\partial v_i}$$

The dynamics of the network are governed by the following equations:

$$\frac{du_{Xi}}{dt} = -\frac{u_{Xi}}{\tau} + S_{Xi},$$

where τ is a network parameter, and the stimulus received from all the neurons (S_{Xi}):

$$S_{Xi} = -A\sum_{j\neq i} V_{Xj} - B\sum_{Y\neq X} V_{Yi} - C(\sum_{Y}\sum_{j} V_{Yj} - N)$$

$$-D\sum_{Y} d_{XY}(V_{Y,i+1} + V_{Y,i-1})$$

To find a solution, we select at random an initial state for the network and let it evolve according to the equations of motion, which were stated above. When energy will reach a steady state iterations will stop. The energy E has many minima and deeper minimum corresponds to the shorter distance.

4 Experiments

For the project we were given a TSP for 10 cities (table 1).

City	X	Y
$\hat{A}(1)$	0.4000	0.4439
$\hat{B}(2)$	0.2439	0.1463
$\hat{C}(3)$	0.1707	0.2293
$\hat{D}(4)$	0.2293	0.7610
$\hat{E}(5)$	0.5171	0.9414
$\hat{F}(6)$	0.8732	0.6536
$\hat{G}(7)$	0.6878	0.5219
$\hat{H}(8)$	0.8488	0.3609
$\hat{I}(9)$	0.6683	0.2536
$\hat{J}(10)$	0.6195	0.2634

Table 1: Cities and their coordinates

Network parameters represented at table 2. Desired total distance and route were taken from project

Parameter	Value
A	500
B	500
C	1000
D	500
τ	1
$u_0(\gamma)$	0.02

Table 2: Cities and their coordinates

requirements. Desired total distance must be equal to $l = 2.6907$ and route must be:

$$\hat{A} \to \hat{D} \to \hat{E} \to \hat{F} \to \hat{G} \to \hat{H} \to \hat{I} \to \hat{J} \to \hat{B} \to \hat{C} \to \hat{A}.$$

Every time we run our network our initial value matrix initialization is random (randomly sampled from uniform distribution) so achieving optimal solution time wont be the same. We report one run, which first desired solution (which had the requested total distance) was achieved in 29 runs(fig 3). Desired city sequence was achieved after 652 runs(fig. 4). Note that city numbering on figures starting from zeros (city 0 equivalent $\hat{A}$)

```
Obtained city sequence: 7=>8=>9=>1=>2=>0=>3=>4=>5=>6=>7
 Final permutation matrix
[[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]]
```

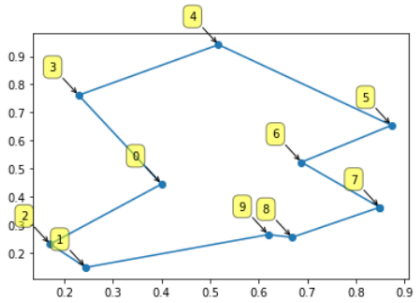

Figure 3: First obtained desired length, but not desired route. **Left**-permutation matrix, **right**-route

```
Obtained city sequence: 0=>3=>4=>5=>6=>7=>8=>9=>1=>2=>0
 Final permutation matrix
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]]
```

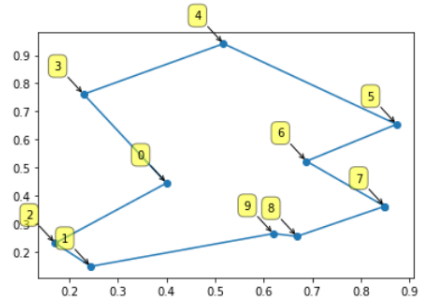

Figure 4: First obtained desired length and route **Left**-permutation matrix, **right**-route

5 Conclusion

We implemented Hopfield Neural network, which successfully gave the desired solution for 10-cities travel salesman problem. Also we discovered, that adding noise more than $0.9u_{start}$ to the starting input potential can influence convergence speed.

References

[1]Hopfield, J.J. and Tank, D.W. Biol. Cybern. (1985) 52: 141. doi:10.1007/BF00339943

[2]Kamgar-Parsi B. and Kamgar-Parsi Behzad Hopfield Model and optimization Problems