IMPLEMENTATION OF TRAVELING SALESMAN'S PROBLEM USING NEURAL NETWORK

FINAL PROJECT REPORT (Fall 2001)

ECE 559 Neural Networks
December 3, 2001

Prof. Daniel Graupe

Ritesh Gandhi

Approximate solution for the Traveling Salesman's Problem Using Continuous Hopfield Network

Ritesh Gandhi

Department of Electrical and Computer Engineering

rgandhi@ece.uic.edu

ABSTRACT

I have proposed an implementation of an algorithm in neural network for an approximate solution for Traveling Salesman's Problem. TSP is a classical example of optimization and constrain satisfaction problem which falls under the family of NP-complete of problems. I have used Continuous Hopefield network to find the solution for the given problem. The algorithm gives near optimal result in most of the cases for upto 20 cities.

PROBLEM

There is a list of cities that are to be visited by a salesman. A salesman starts from a city and come back to the same city after visiting all the cities. Here the objective is to find the path, which follows following constrains

1) Salesman has to visit each city. He should not leave any city unvisited.
2) Each city should be visited only one time.
3) The distance that he travels till he returns back to the city he has started should be minimum.

INTRODUCTION

The traveling salesman problem (TSP) is well known in optimization. The TSP problem is NP-complete problem. There is no algorithm for this problem, which gives a perfect solution. Thus any algorithm for this problem is going to be impractical with certain examples.

Here we assume that we are given n cities, and a non-negative integer distance Dij between any two cities i and j. We try to find the tour for the salesman that best fits the above-mentioned criterion.

There are various neural network algorithm that can be used to try to solve such constrain satisfaction problems. Most solution have used one of the following methods

- Hopfield Network
- Kohonen Self-organizing map
- Genetic Algorithm

Here an approximate solution is found for TSP using Hopfield network.

HOPFIELD NETWORK

Hopfield network is a dynamic network, which iterates to converge from an arbitrary input state. The Hopfield Network works as minimizing an energy function.

The Hopfield net is fully connected network. It is a weighted network where the output of the network is fed back and there are weights to each of this link. The fully connected Hopfield network is shown in following figure.

Here we use n^2 neurons in the network, where n is the total number of cities. The neurons here have a threshold and step-function. The inputs are given to the weighted input node. The network then calculates the output and then based on Energy function and weight update function, converges to the stable solution after few iteration. The most important task on hand is to find an appropriate connection weight. It should be such that invalid tours should be prevented and valid tours should be preferred.

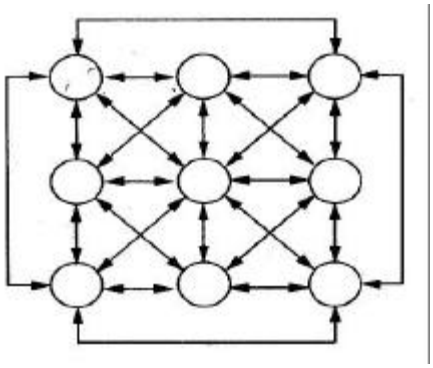

Figure : Fully Connected Hopfield Network for TSP for 3 cities.

The output result of TSP can be represented as following. The example here is for 4 cities. The 4 cities TSP need 16 neurons.

	#1	#2	#3	#4
C1	0	1	0	0
C2	1	0	0	0
C3	0	0	0	1
C4	0	0	1	0

Figure : Tour Matrix obtained as the output of the network.

The corresponding visiting route, in the above example is

City2→ City1→City4→City3→City2

So the total traveling distance is

$D = D_{21} + D_{14} + D_{43} + D_{32}$.

NETWORK INPUTS

The inputs to the network are chosen arbitrarily. The initial state of the network is thus not fixed and is not biased against any particular route. If as a consequence of the choice of the inputs, the activation works out to give outputs that add up to the number of cities, and initial solution for the problem, a legal tour will result. A problem may also arise that the network will get stuck to a local minimum. To avoid such an occurrence, random noise is generated and added.

Also there are inputs that are taken from user. The user is asked to input the number of cities he want to travel and the distance between those cities which are used to generate the distance matrix.

Distance matrix in n*n square matrix whose principal diagonal is zero. The

figure below shows a typical distance matrix for 4 cities.

	C1	C2	C3	C4
C1	0	10	18	15
C2	10	0	13	26
C3	18	13	0	23
C4	15	26	23	0

Figure : Distance Matrix generated after getting inputs from user

Here distance, for example, between city C1 and city C3 is 18 and distance between a city to itself is zero.

ENERGY FUNCTION

The Hopfield network for the application of the neural network can be best understood by the energy function. The energy function that is developed by Hopfield and Tank is used for the project. The energy function has various hollows that represent the patterns stored in the network. An unknown input pattern represents a particular point in the energy landscape and the pattern iterates its way to a solution, the point moves through the landscape towards one of the hollows. The iteration is carried on till some fixed number of time or till the stable state is reached.

The energy function used should satisfy the following criterions

- The energy function should be able to lead to a stable combination matrix.
- The energy function should lead to the shortest traveling path.

The energy function used for the hopfield neural network is

$$E = A_1 \Sigma_i \Sigma_k \Sigma_{j\neq k} X_{ik} X_{ij} + A_2 \Sigma_i \Sigma_k \Sigma_{j\neq k} X_{ki} X_{ji} + A_3 [(\Sigma_i \Sigma_k X_{ik}) - n]^2 + A_4 \Sigma_k \Sigma_{j\neq k} \Sigma_i d_{kj} X_{ki}(X_{j,i+1} + X_{j,i-1})$$

Here A_1, A_2, A_3, A_4 are positive integers, the setting of these constants are critical for the performance of Hopfield network. Xij is the variable to denote the fact that city i is the jth city visited in a tour. Thus Xij is the output of the jth neuron in the array of neurons corresponding to the ith city. We have n^2 such variable and their value will finally be 0 or 1 or very close to 0 or 1.

The Energy function can be analyzed as follows

- ***ROW CONSTRAINT***: ($A_1 \Sigma_i \Sigma_k \Sigma_{j\neq k} X_{ik} X_{ij}$) In the energy function the first triple sum is zero if and only if there is only one "1" in each order column. Thus this takes care that no two or more cities are in same travel order. i.e. no two cities are visited simultaneously.
- ***COLUMN CONSTRAINT:*** ($A_2 \Sigma_i \Sigma_k \Sigma_{j\neq k} X_{ki} X_{ji}$) In the energy function the first triple sum is zero if and only if there is only one city appears in each order column. thus this takes care that each city is visited only once.

- ***TOTAL NUMBER OF "1" CONSTRAINT:*** (A_3 $[(\Sigma_i \Sigma_k X_{ik})-n]^2$)The third triple sum is zero if and only if there are only N number of 1 appearing in the whole n*n matrix. Thus this takes into care that all cities are visited.
- The first three summation are set up to satisfy the condition 1, which is necessary to produce a legal traveling path.
- ***SHORTEST DISTANCE CONSTRAINT:***[A_4 Σ_k $\Sigma_{j\neq k}$ Σ_i d_{kj} $X_{ki}(X_{j,i+1} + X_{j,i-1})$] The forth triple summation provides the constrain for the shortest path. d_{ij} is the distance between city i and city j. The value of this term is minimum when the total distance traveled is shortest.]
- The value of A_4 is important to decide between the time taken to converge and the optimality of the solution. If the value of A4 is low it takes long time for the NN to converge but it gives solution nearer to the optimal solution but if the value of A_4 is high the network converges fast but the solution may not be optimal.

WEIGHT MATRIX

The network here is fully connected with feedback and there are n^2 neurons, thus the weight matrix will be a square matrix of n^2*n^2 elements.

According to the Energy function the weight matrix can be set up as follows

$$\mathbf{W_{ik,\,lj} = -A_1\,\delta_{il}\,(1-\delta_{kj}) - A_2\,\delta_{kj}\,(1-\delta_{jl}) - A_3 - A_4\,d_{jl}\,(\delta_{j,\,k+1} + \delta_{j,\,k-1})}$$

Here the value of constants A_1, A_2, A_3, A_4 is same as we have it in the Energy function. Weights are also updated keeping into mind various constraints to give a valid tour with minimum cost of travel. In this context, the Kronecker delta function (δ) is used to facilitate simple notation.

The weight function can be analyzed as follows

- The neuron whose weight is updates is referred with two subscripts, one for the city it refers to and the other for the order of the city in the tour. Therefore, an element of the weight matrix for a connection between two neurons needs to have four subscript, with a comma after two of the subscripts.
- The negative signs indicate inhibition through the lateral connections in a row or a column.
- The Kronecker delta function has two arguments (two subscripts of the symbol δ). By definition δ_{ik} has value 1 if i =k, and 0 if i$\neq$ k.
- The first term gives the row constraint, thus taking care that no two cities are updated simultaneously.
- The second term gives the column constraint, thus taking care that no city is visited more than once.
- The third term here is for global inhibitation
- The fourth term takes care of the minimum distance covered.

ACTIVATION FUNCTION

The activation function also follows various constraints to get a valid path. Hence the activation function can be defined as follows.

$$a_{ij} = \Delta t(T_1 + T_2 + T_3 + T_4 + T_5)$$

$$T_1 = -a_{ij} / t$$

$$T_2 = -A_1 S_i X_{ik}$$

$$T_3 = -A_2 S_i X_{ik}$$

$$T_4 = -A_3 (S_i S_k S_{ik} - m)$$

$$T_5 = - A_4 S_k d_{ik} (X_{k, j+1} + X_{k, j-1})$$

- We denote the activation of the neuron in the ith row and jth column by aij, and the output is denoted by xij.
- A time constant τ, is also used. The value of τ is taken as
- A constant m is also another parameter used. The value of m is
- The first term in activation function is decreasing on each iteration
- The second, third, fourth and the fifth term give the constraints for the valid tour.

The activation is updated as

$$a_{ij(new)} = a_{ij(old)} + \Delta a_{ij}$$

OUTPUT FUNCTION

This a continuous hopfield network with the following output function

$$X_{ij} = (1 + \tanh (\lambda a_{ij})) / 2$$

- Here Xij is the output of the neuron.
- The hyperbolic tangent function gives an output as shown in the figure below.
- The value of λ determines the slope of the function. Here the value of λ is 3.

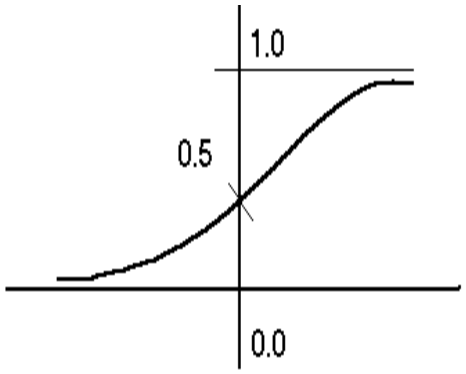

Figure : Variation of Output function

- Ideally we want output either 1 or 0. But the hyperbolic tangent function gives a real number and we settle at a value that is very close to desired result, for example, 0.956 instead of 1 or say 0.0078 instead of 0.

PROGRAM DETAILS

The algorithm is coded in C++ for the Hopfield network operation for the traveling salesman's problem. The present program is for maximum 20 cities but this program can be very easily extended for more number of cities.
The following is a listing of the characteristics of the C++ program along with definitions and/or functions.

- The number of cities and the distances between the cities are solicited from the user.
- The distance is taken as integer values.
- A neuron corresponds to each combination of a city and its order in the tour. The ith city visited in the order j, is the neuron corresponding to the element j + i*n, in the array for neurons. Here n is the number of cities. The i and the j vary from 0 to n-1. there are n^2 neurons.
- mtrx is the weight matrix and it gives the weights on the connection between the neurons. It is a square matrix of order n^2.
- An input vector is generated at random in the function main ( ), and is later referred to as input.
- getdistance( ) : takes the distances between corresponding cities from the user.
- Asgninpt( ) : assign the intial input to the network, and this are randomly generated
- findpath( ): this function finds the final path that is optimal or near optimal. It iterates and the final state of the network is set in such a way that all the constraint of the network is fulfilled.
- findtour( ): it generates a Tour Matrix and the exact route of travel.
- calcdist( ): calculates the total distance of the tour based on the tour generated by the function findtour( ).

PARAMETERS

The parameter settings in the Hopfield network are critical to the performance of the Network. The various parameter used and their intial value set are as follows

A1 : 0.5
A2 : 0.5
A3 : 0.2
A4 : 0.5
λ : 3.0
τ : 1.0
M : 15

OUTPUT RESULT OF TSP PROGRAM

The attached result shows the simulation using 4, 8, 10, 15, 20 cities. The traveling paths generated are shown in the form of the matrices, which are in the "output.txt" file.

Hopfield neural network is efficient and it can converge to stable states in hundreds times iterations. The state values are analog to facilitate the finite differential calculation.

The output.txt file first gives the inputs that are taken from the user. i.e. the number of cities and their distance in the form of distance matrix. Then for those cities the output that is generated is printed in the form of Tour Matrix, Tour Route and Total Distance Traveled. The solution is optimal or near optimal.

DISCUSSION OF RESULT AND COMPARISIONS

TSP, one of the most famous NP-complete problems, has been simulated using Hopfield Network. The simulation is satisfactory.

The result attached along with the code are for 4, 8, 10, 15, 20 cities respectively. The number of iteration required to converge the network in each case can be summarized as follows.

4 cities	: 89 iterations
8 cities	: 355 iterations
10 cities	: 586 iterations
15 cities	: 1021 iterations
20 cities	: 2433 iterations

- The result above shows that as the number of cities increases the number of iteration required increases sharply. The increase is not a linear increase.
- One more thing that was noticed is that the number of iteration required for the convergence did not remain same for any particular city. For example for 4 cities the network usually converged after 80 to 110 iterations, but on few occasions it took around 60 iterations while in few cases it didn't converge at all or took more than 250 iterations. This is because the initial network state is randomly generated. This may sometimes result to no convergence also.
- Many times the result converges to local minimum instead instead of global minimum. To avoid this a random weights were added to the initial inputs.
- The algorithm developed in non-deterministic. Thus it does not promise an optimal or near optimal solution every time. Though it does gives near optimal solution in most of the cases, it may fail to converge and give a correct solution.
- Many times when the energy of the system was calculated, it was found to increase instead of decreasing. Thus the algorithm failed in few cases. This again was the consequence of the random initial state of the network.
- In **94 %** of test cases the algorithm converged, while in **4 %** algorithm failed to converge and in remaining **2%** the energy of the system increased instead of decreasing.

There are various advantages of using Hopfield network though I had many other approaches like Kohonen Network and Genetic Algorithm approach.

- Hopfield neural network setup is very optimal for the solution of TSP. It can be easily used for the optimization problems like that of TSP.
- It gives very accurate result due to very powerful and complete Energy equation developed by Hopfield and Tank.
- The approach is much faster than Kohonen as the number of iteration required to get the solution is less.
- The result obtained is much more near optimal than compared to Genetic Algorithm approach as in genetic algorithm it is more like trial error and chances to get the optimal solution is less.
- This neural network approach is very fast compared to standard programing techniques used for TSP solution.
- With very few changes this algorithm can be modified to get the approximate solution for many other NP-complete problems.

PROBLEMS FACED

- The understanding of Energy function initially was difficult. As this Hopfield network is not a usual Character recognition problem, which was solved initially. But once I went through the literature and papers, it became very clear.
- The understanding of output and activation and weight update functions also was initially difficult, but became very clear later.
- The setting for various parameter values like A_1, A_2, A_3, A_4, λ, τ, m, etc was a challenge. The best value was chosen by trial and error. Improvement is still possible for this parameters value.
- Many times the algorithm converged to local minima instead of global minimum. This problem was mostly resolved by adding a random noise to the initial inputs of the system.
- The testing of algorithm gets difficult as the number of cities increase. Though there are few software and programs available for the testing, none of them guarantees the optimal solution each time. So an approximation was made during the testing of the algorithm.

FURTHER SCOPE OF DEVELOPMENTS

- The network here developed does not always give optimal solution though in most cases it's near optimal. Few more changes or improvement can be made to energy function along with other functions like weight updating function and activation function to get better answer.
- Various values of constants (i.e. A, B, C, D) can be tried to get optimal or near optimal result in the present algorithm.
- Even if one of the distances between the cities is wrong the network has to start form the very first stage. Some way or method needs to develop so that this error can be handled.
- If we want to add or delete a city, the network has to be again started from the initial state with the required changes. Some equations can be developed so that these changes can be incorporated.
- Optimization of the result and cooling can be done on the network, which will improve the result and the convergence of the result.
- The algorithm can be modified for solving other NP-complete problems.

REFERENCES

- Graupe, Daniel 1997. Principles of Artificial Neural Networks. Advanced Series on Circuits and Systems – Vol.3
- Hopfield, J.J and Tank, D.W., neural Computation of Decision in Optimizations Problems, Biol. Cybern. No. 52
- Roa & Roa. C++ Neural Network and Fuzzy Logic. Second Edition.
- Laurene V. Fausett, Fundamentals of Neural Networks. Architectures. Algorithms and Applications.
- Timothy Masters, Practical Neural Network Recepies in C++.