Application of Self Organizing Maps to Travelling-Salesman Problem

Aditya Shankar Garg
Department of Mechanical Engineering
Indian Institute of Technology, Delhi
me1190770@iitd.ac.in

Ishan Singh
Department of Mechanical Engineering
Indian Institute of Technology, Delhi
me11902520@iitd.ac.in

Gurpinder Singh
Department of Civil Engineering
Indian Institute of Technology Delhi
ce1190243@iitd.ac.in

Abstract

This work shows how a modified Kohonen Self-Organizing Map with one dimensional neighborhood is used to approach the symmetrical Traveling Salesperson Problem. Solution generated by the Kohonen network is improved by the 2opt algorithm.

1 Introduction:

1.1 Travelling Salesman Problem

Given the graph G = (V, E) of cities we have to find the route that has the minimum cost of travel and visits each of the city exactly once and ends at the starting city. In a graph of N- cities the total number of paths (search space) will be N! Hence it is a computationally costly problem to solve (NP Hard). While there are other conventional solutions for the problem using dynamic programming and graph algorithms, our main focus is to design and train a self-organizing map that finds optimal/ sub-optimal routes for the graph.

1.2 Self Organising Maps

The Self Organising Maps or SOMs are a type of artificial neural network architecture that uses unsupervised, competitive learning to produce a low dimensional, discretized representation of the input space of the training samples (map) and therefore is a method to do dimensionality reduction. It is differing from other ANN as it applies competitive learning as oppose to error correction learning, and in the sense that they use a neighborhood function to preserve the topological properties of the input space. In SOM we majorly use two kinds of competitive learning algorithms, Winner Takes All (WTA) and Winner Takes Most(WTM).

2 Methodology:

Basically, SOMs work on competitive learning algorithms. For instance, if we consider the Winner Takes All Algorithm, we first randomize the neuron weights then we calculate the distance of the input vector from each of the weights. The neuron whose distance from the input is the minimal is the winner neuron. Then we only modify the weights of the winning neuron to the point that is closest in the input data ($w_i \leftarrow w_i + \eta(x - w_i)$), where w_i are the weights of the winning neuron, η is the learning rate and, x is the current input vector. In the Winner Takes Most algorithm (WTM) we not only update the weights of the winning neuron but also update the weights for the neighbor neuron that may or may not be activated. Due to increase in the updating at each epoch learning takes place faster and the algorithm converges faster than the WTA. But an important thing to note is that in WTM all the neighbors of the winner aren't updated equally. We define a close neighborhood of the winning neuron and then update the weights in that neighborhood only. ($\delta(w_i, x) \leq \lambda,\ w_i \leftarrow w_i + \eta\, N(i,x)(x - w_i)$, where $N(i,x)$ corresponds to a neighborhood function which can be in different forms like gaussian neighborhood or $N(i,x) = 1$ if $\delta(w_i, x) \leq \lambda$ else $N(i,x) = 0$) we repeat this procedure at each epoch reducing the values of η and λ.

To use this network to solve the travelling salesman problem we need to understand how to modify our neighborhood function. Instead of a grid we modify the architecture of the network to a circular array/ list of neurons in which each neuron will be only conscious of the neurons in front and behind. Our self-organizing map will behave as an elastic ring that gets close to the cities while also trying to minimize its own perimeter. Note at each step we epoch we update the learning rate and the neighborhood using the following equations, $\eta_{t+1} = \gamma_\eta \cdot \eta_t$, $\lambda_{t+1} = \gamma_\lambda \cdot \lambda_t$

The parameters γ_η , γ_λ are the decay rates of the learning rate and neighbor distance parameter respectively. These decay rates will ensure that our algorithm finally converges to a solution.

To obtain the route from the self-organizing map it is necessary that we assign a winner neuron to each city, so for that we traverse the ring starting from any point and sort the cities by the order of appearance of their winner neuron in the ring. It can also happen in our solution that several cities map to a single neuron, in that case we can take any possible order for traversal. After we have found the route, we will extract it at try to implement the 2-OPT algorithm to optimize our solution.

The pseudo-code for our algorithm for generating the route looks something like this:

```
Iterate of the number of training cycles

    Iterate through all the neurons

        find the nearest city corresponding to the neuron

        if city not assigned to another neuron:
            assign the city to the current neuron

        else if city assigned to another neuron:
            delete the current neuron


    Iterate through all the cities

        if current city not assigned:

            create a new neuron
            assign to the current city

            find the nearest neuron to the current city

            if length (new-neuron, neuron) > length (neuron, new-neuron):
                insert new- neuron after
            else:
                insert new- neuron before
```

3 Implementation:

Our first goal is to extract the data from the TSPLIB file and run the network to obtain the shortest route. Then we will try to normalize our data as the coordinates can be large at times so to avoid any kind of overflow errors or computational complexity, we scale our data in the range of (0, 1]. Then feed the data to our neural network.

The functionalities of implementation are collected in several files (.ipynb format) in the notebooks directory :

- extraction_preprocessing_visualisation.ipynb – contains the functions that allow us to read files from the tsp format and store it in the pandas data-frame object. We have also included the visualization of the data using the seaborn and matplotlib modules.
- helper_functions_distance_calculator.ipynb – contains the functions that implement the Euclidean distance, the cost of traversing in a route in a particular order and selecting the closest city/ neuron from the neighbors
- neuron.ipynb – contains the code for generation of a new network of a specified size, the code for implementing a gaussian neighborhood function for the network and also the code for implementation of a self-organizing map.
- plotting_capabilities.ipynb – contains the code for the functions that plot the route and network at different epochs.
- 2opt_and_graphs.ipynb – contains the code for the two opt algorithms using the C backend that can be used to refine the route generated using the self-organizing map also has the implementation of the graph data structure in form of an adjacency matrix that was not used but could have been

We have majorly used the Pandas, NumPy packages for storing the data, computations and vectorization respectively. We will be using Matplotlib and Seaborn for plotting capabilities. These all-necessary tools were included with the Anaconda distribution of Python.

We finished off the work left on the implementation of the 2-OPT algorithm, also we have implemented the plotting capabilities, to store the state of the network at every 1000 iterations. We have removed some bugs from the earlier code.

4 Evaluation and Results:

To evaluate our model, we need to define certain metrics:

- execution time, the total time taken by the technique to find a solution to the given dataset
- quality of solution, the optimal route can be found out using the conventional ways and the prior knowledge, if our solution is in an acceptable range from the length of the optimal solution then it is a good sub-optimal solution.

We will use different parameters to train the model, which need to be figured out via experimenting and parametrization. Initially we will try to use parameters from the following list:

- the initial size of the neuron ring (population size) will be considered as a random integer in the range [1 11) times the number of cities. We could have found out a fixed parameter for this job using the grid search method, it can be left for future improvement.
- The initial learning rate will be taken as 0.9 or 0.8 and will decay at a rate of 0.99995 or 0.99997.
- The initial neighborhood of the number of cities will be decayed by a factor of 0.9995.

The used datasets from the TSPLIB library are:

- Qatar, 194 cities, optimal tour length of 9352 units
- Uruguay, 734 cities, optimal tour length of 79114 units

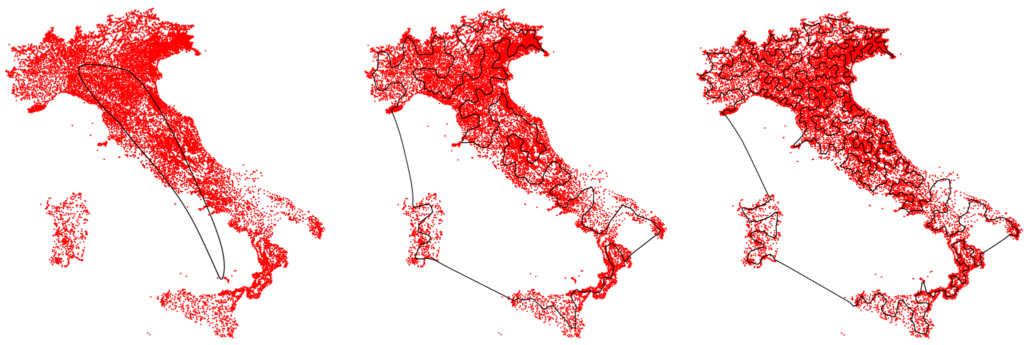
- Finland, 10639 cities, optimal tour length of 520527 units
- Italy, 16862 cities, optimal tour length of 557315 units

It was found out that taking a decay rate of around 0.8, 0.95 and a learning rate of around 0.9997 or higher results into very fast decay of the learning rate. Hence to avoid the problem of underfitting we have decided to take the decay rates of the order of 0.9997 or higher.

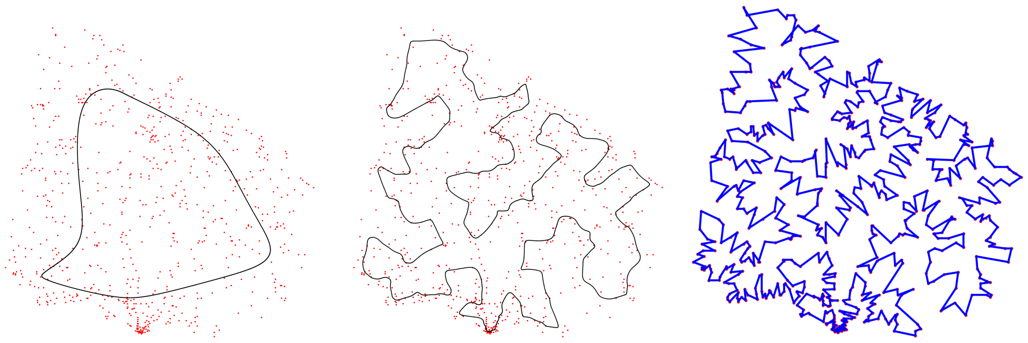
Now to evaluate the performance of the implementation we have decided to use the parameter of distance primarily:

SI No.	City	Iterations	Length	Quality
1.	Qatar	23,022/ 1,00,000	10223.890	0.093230
2.	Italy	99,000/ 1,00,000	723201.75	0.297653
3.	Uruguay	26,021/ 1,00,000	85068.060	0.075259

It can be seen that the number of iterations it takes to reach the optimal solution varies with the size of the dataset, this can also be stated in terms of the time of execution of the process. One way to explain this increase is due the complexity of searching for the winner neuron in a big dataset takes time. We tried to compute the routes using other parameters too and ended with the routes with quality always less than 0.35.



the above three figures demonstrate the state of the network for the Italy dataset at three distinct number of iterations, it can be noticed how the network stretches like a ring to fit on to the dataset while at the same time minimizing the route distance. We initially didn't take the number of neurons same as the number of cities that was done majorly to exploit the fact that there may be local better ways inside the data, so for that purpose number of neurons were increased by the factor of a random integer in between 1 to 10.



It was seen that the 2-OPT algorithm didn't improve the route by much and instead took a lot of time for its execution even after improving it using the C backend so it was concluded that a sub-optimal solution finding using an SOM was much efficient that refining that solution using a optimiser such as a 2-OPT algorithm, however this approach seemed to be novel and efficient. The quality of the solution generated by the SOM can be improved even further if instead of the gaussian neighbourhood we tried to implement a Kronecker-Delta based neighbourhood that updates only the neurons that are within a given radius of the winner.

5 Conclusion:

The network can be used to find sub-optimal solution with high quality, although this can be improved with better parameterisation, the current solution is performs well and is time efficient unlike the traditional ways to solve the problem which take huge time for large datasets. If we would have used the parameterisation technique to search for better parameters then even though our quality would have improved the time taken in the overall procedure would have increased exponentially.

Statement of Contribution

Aditya Shankar Garg – wrote the code for neuron.ipynb, refined the SOM function to counter the overflow error (SOM-II.ipynb) and wrote sections 2, 4. Gurpinder Singh– wrote the code for data handling, pre-processing and visualisation and the two_opt.ipynb and contributed to the writing of section 1. Ishan Singh – wrote the code for the distance and helper functions, the plotting capabilities and helped in neuron.ipynb and wrote section 3, 5

References

[1] L. Brocki, "Kohonen self-organizing map for the traveling salesperson," in Traveling Salesperson Problem, Recent Advances in Mechatronics, pp. 116– 119, 2010.

[2] Papadimitriou, C.H., "The Euclidean Traveling Salesman Problem is NP-complete", Theoretical Computer Science, 4, 1978, pp. 237-244.

[3] X. Xu, Z. Jia, J. Ma and J. Wang, "A Self-Organizing Map Algorithm for the Traveling Salesman Problem," 2008 Fourth International Conference on Natural Computation, Jinan, China, 2008, pp.431-435, doi: 10.1109/ICNC.2008.569.

[4] Kohonen T. (2001), Self-Organizing Maps, Springer, Berlin

[5] B. Angeniol, G. D. L. C. Vaubois, and J.-Y. L. Texier, "Self-organizing fea-ture maps and the travelling salesman problem," Neural Networks, vol. 1, no. 4, pp. 289–293, 1988.

[6] K. L. Hoffman, M. Padberg, and G. Rinaldi, “Traveling salesman problem,” in Encyclopedia of operations research and management science, pp. 1573– 1578, Springer, 2013.