Deep Learning for Travelling Salesman Problem

Aditi Dandekar
aadandek@ncsu.edu

Hitarth Shah
hshah4@ncsu.edu

Shreya Parikh
scparik2@ncsu.edu

***Abstract*—The travelling salesman problem (TSP) is a combinatorial optimization and integer programming problem that calls for the determination of the optimal set of routes to be performed by a salesman to serve a given set of customers.We present an end-to-end framework for solving the problem using reinforcement learning. In this approach, we train a single policy model that finds near-optimal solutions for a broad range of problem instances of similar size, only by observing the reward signals and following feasibility rules. We consider a parameterized stochastic policy, and by applying a policy gradient algorithm to optimize its parameters, the trained model produces the solution as a sequence of consecutive actions in real time, without the need to re-train for every new problem instance.Our proposed framework can be applied to other variants of the TSP such as the stochastic TSP, and has the potential to be applied more generally to combinatorial optimization problems.**

I. Introduction

The objective of the project is to solve classic combinatorial optimization problem - Travelling Salesman Problem - using deep learning algorithm. Problem Statement of the Travelling Salesman Problem is "Given a list of cities and distances between each pair of cities, what is the shortest possible route that visits each city and returns to the returns to the origin city?". It is a NP-Hard problem in combinatorial optimization. NP-Hardness in computational complexity theory is the defining property of a class of problems that are informally as hard as the hardest problem in NP. The Travelling Salesman Problem (TSP) was first formulated in 1930 and it is one of the most intensively studied problems. The TSP problem has several applications in fields of planning, logistics, manufacture of microchips and DNA sequencing. In these examples cities are in fact customers, soldering points, or DNA fragments and the distance represents travelling time or cost, or a similarity measure between DNA fragments. Additional constraints in the form of capacity and time windows are introduced which gives rise to many generalizations of the TSP problem. Some of them are the Vehicle Routing Problem and the Set TSP problem. In this project, we develop a framework with capability to solve the TSP problem. Fig. 1 shows a solution of a TSP problem where black line is the shortest possible route connecting all red dots.

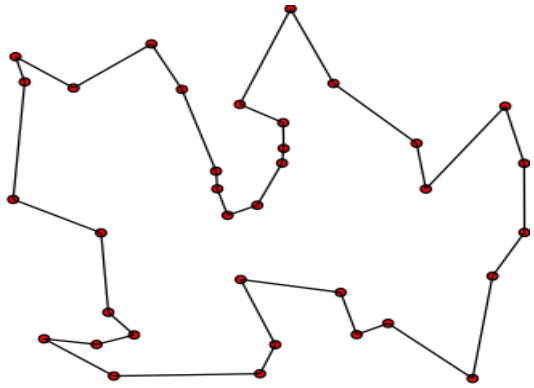

Fig. 1. TSP Problem

II. Related Work

One of the papers presented an end to end framework for solving the Vehicle Routing Problem (VRP) using reinforcement learning [4]. They developed a framework develop a framework with the capability of solving a wide variety of combinatorial optimization problems using Reinforcement Learning (RL) [4] and show how it can be applied to solve the VRP. They proposed a model had a recurrent neural network decoder coupled with the attention mechanism. At each time step, the embeddings of the static elements are the input to the RNN decoder, and the output of the RNN and the dynamic element embeddings are fed into an attention mechanism, which forms a distribution over the feasible destinations that can be chosen at the next decision point. They obtained split delivery without any hand engineering and no extra cost. The drawback of this solution is that the method scales well as the problem size increases, and it has superior performance with competitive solution-time. [4]

The paper titled - Learning Combinatorial Optimization Algorithms over Graphs [3] - talks about good heuristics or approximations algorithms for NP-Hard combinatorial problems. The learned greedy policy behaves like a meta-algorithm that incrementally constructs a solution, and the action is determined by the output of a graph embedding network capturing the current state of the solution.The paper applies this strategy on different problems like Minimum Vertex Cover and Maximum Cut. In this project we need a neural network graph structure which learns any random graph. An architecture which suits these needs is proposed in this paper using an algorithm called structure2vec [2]. Thus, using this technique Q() function is able to find fixed-size graph embedding. [3]

III. Algorithm

Our approach of this project is to solve the combinatorial optimization problem - Travelling Salesman Problem - using Reinforcement Learning on Graphs. Neural Networks are used to learn embeddings of random graphs and Reinforcement Learning is used to iteratively build solutions. The Technology

Tool stack which is used to develop this project is given in Table.

Libraries	Function Names
numpy	random, dumping array to npy, ones, median, tolist
PyTorch	Define Neural Network architecture model
matplot	For plotting the graph
Google Colab	Execution Environment

TABLE I
TOOLBOX

Reinforcement learning [1] is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. It is one of the three basic Machine Learning paradigms, alongside supervised learning and unsupervised learning. Reinforcement Learning [1] is stated as Markov Decision Process (MDP) because this context is helpful in dynamic programming. Most common framing of Reinforcement Learning is given in Fig. 2: From the above Fig. 2, Agent is

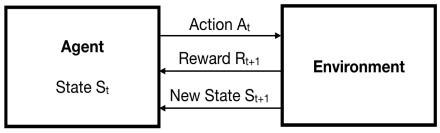

Fig. 2. Reinforcement Learning

in State 'S'. Agent can observe the Environment and take an action 'A' on it. Based on the Action Agent receives a Reward 'R'. The reward can be positive gain or negative gain. This also lands agent in a new State 'S'. Thus, the aim of Reinforcement Learning is to training the Agent in such a way that it takes the Action which maximizes its Reward. In our project, agent is interested in finding short tours. Environment is considered to be fully observed and trivial. Agent makes a deterministic move along the graph and observes the distance travelled. The 'State' of the agent is depicted by (Fig. 3):

- Entire graph connecting all the points (Euclidean distance from each point to every other point)
- Coordinates of the point on the plane
- Partial solution built so far while execution

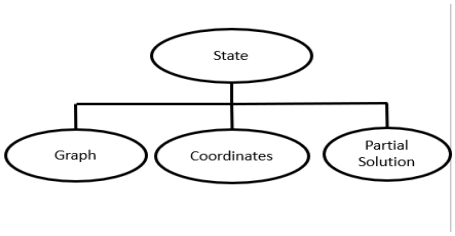

Fig. 3. State

Neural Network is a series of algorithms to recognize underlying relationships in a set of data through a process which mimics the human brain. Neural Network are made of connected units or nodes called neurons. Neurons are arranged in multiple layers. The layer which receives external data is the input layer. The layer which produces the end result is the output layer. Between them, are zero or more layers called the hidden layers. This architecture is shown in Fig. 4

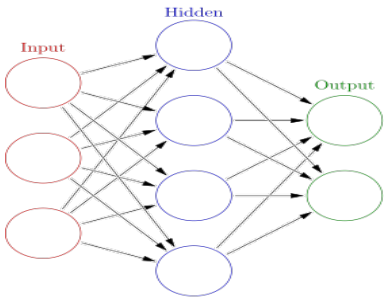

Fig. 4. Neural Network

The Neural Network architecture which we have used for this project consists of 8 Linear layers. Input to the network is a 5 dimensional tensor. Each of this tensor corresponds to a point on the graph. Tensor of the node stores:

- 'x' coordinate of the point
- 'y' coordinate of the point
- binaryValue to determine if the point is present in the solution developed so far
- binaryValue to determine if the point is the first node in the solution
- binaryValue to determine if the point is the last node in the solution

Some hyperparameters set for this Neural Network are:

- ReLU is the activation function
- Mean Square Error is the loss function
- Adam Optimizer

Q-Learning is a off-policy Reinforcement Learning algorithm which seeks to find the best action to take given the current state. It is called off-policy because Q-Learning allows to take random actions and hence no policy is needed. For Q-Learning, the agent learns an action-value function Q(s,a) i.e for given state 's' and action 'a', the algorithm returns the expected cumulative reward which the agent the agent will obtain at the end of one instance of TSP also known as episode. Thus, the agent picks and action which maximizes the estimated expected return from the function Q(s,a).

While training the agent based on Q-Learning, its experience is stored in the memory. Experience of the agent is stored in the following format (Fig. 5:

- State - current state of the agent
- Action - action which the agent selects by observing the Environment
- Reward - Reward points which the agent received on taking this action. In this project reward refers to the Euclidean distance travelled by the agent. The reward points are considered negative as the aim is to maximize the reward.

- NextState - Next State in which the agent lands on taking this action

Fig. 5. Experience

A random batch of experiences are picked from the memory and targets are defined for each of these experiences. Target is identifying for given state 's' if action 'a' is taken, then from which action from the next state will give the best reward. Thus, it refers to recursively calling Q() function to estimate final tour length for a given experience.

IV. Experiment, Results and Comparative Analysis

a) Dataset: In this experiment, the algorithm will generate random graphs with number of nodes assigned by the user,that is, 10, 20, 50,etc. These graphs generated are fully connected graphs. The nodes are placed uniformly at random on the [0,10]X[0,10] square and the edge weights are simply all the pairwise Euclidean distances between the nodes.

b) Experimental Setup: The function of Q-Learning is randomly initialized. The epsilon value is set to an initial value close to 1 with an appropriate decay to limit the randomness over episodes. In order to store the experiences(state – action – reward) from Q-Learning, the memory capacity was set to 10000. For convergence purposes, the number of episodes was set to 3000.

c) Results: From the experiment, it is observed that the decisions made by the model become better. At the beginning of the experiment, the decisions were uniformly random while towards the end of the training it is observed that decisions are better. Looking at the decisions, we can say that the Q-Learning Function got better. In Fig.6, we can see the moving average of the path generated over the course of training for 10 nodes. While in Fig.7, we can see the moving average of the path generated over the course of training for 50 nodes. Using these two graphs, we can see the decrease in average path length as the number of iterations increases. The moving average of total path distance is inversely proportional to the number of episodes.

Fig.8 and Fig.9 are graphs generated randomly and using model respectively. Using these two graphs, we can observe the difference between the two. The graph that is generated randomly in fig.8 looks like an unstructured and messy graph

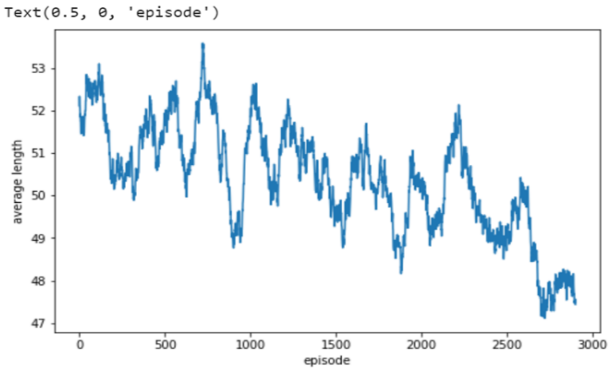

Fig. 6. Moving Average of the Total Distance over the Course of Training for 10 Nodes

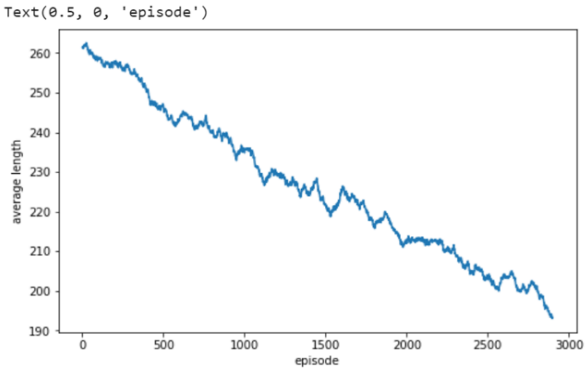

Fig. 7. Moving Average of the Total Distance over the Course of Training for 50 Nodes

while the graph generated using the model in fig.9is more structured and clear. Also, the total path length of random graph is more than the total path length of model graph. There are some examples in which the agent is failing and performing like random strategy. But we can see in the above example, that the learning agent does not always fail. Also, we can observe a pattern where agent is preferring a path which means the agent is learning.

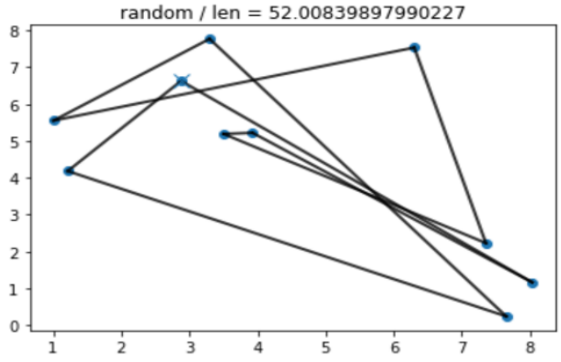

Fig. 8. Random Path Generated

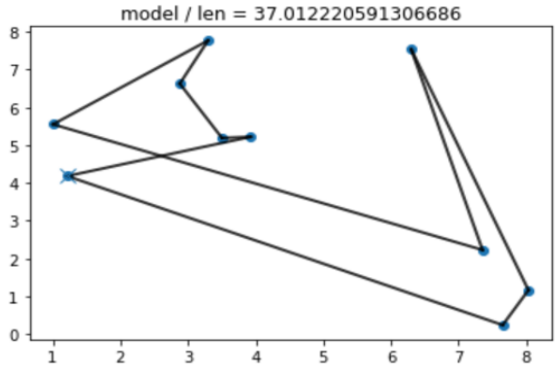

Fig. 9. Path generated by Learning Agent

d) Comparative Analysis: The algorithm used in this experiment is compared to two other algorithms which are used to solve the travelling salesman problem. The two algorithms used are nearest neighbors algorithm and Google ORtools. The nearest neighbour algorithm was one of the first algorithms used to solve the travelling salesman problem approximately because of it's simplicity. Google ORtools is one of the most efficient method to solve the travelling salesman problem.
The table II shows the comparison between the learning algorithm with nearest neighbors algorithm and Google ORtools. To compare these algorithms, each algorithm gives the shortest total path for the graph generated for number of nodes 10,20,50. The results of nearest neighbors and Google ORtools are much better than the Q-Learning Algorithm. Another observation, Google ORtools results are better than Nearest Neighbor algorithm. While Q-Learning is nowhere close to the performance of TSP solvers, the results we found are still amazing. This experiment was to prove that we can solve TSP using Deep Learning Algorithm and not to improve the performance of the algorithm.

TABLE II
COMPARATIVE ANALYSIS

Number of Nodes	Q-Learning	Nearest Neighbors	Google ORTools
10	37.01	25.5	19
20	73.15	40.11	27
50	208.85	58.24	23

V. CONCLUSION

This method is quite appealing since the only requirement is a verifier to find feasible solutions and also a reward signal to demonstrate how well the policy is working. Once the trained policy is available, it can be used many times, without needing to re-train for new problems as long as they are generated from the training distribution. The fact that we can solve similar-sized instances without retraining for every new instance makes it easy to deploy our method in practice. In summary, we expect that the proposed architecture has significant potential to be used in real-world problems with further improvements and extensions that incorporate other realistic constraints.

REFERENCES

[1] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602. 2013.
[2] Dai, Hanjun, Bo Dai, and Le Song. "Discriminative embeddings of latent variable models for structured data." ICML. 2016.
[3] Khalil, Elias, et al. "Learning combinatorial optimization algorithms over graphs." NeurIPS. 2017.
[4] Mohammadreza Nazari, Afshin Oroojlooy, Martin Taká˘c and Lawrence V. Snyder. "Reinforcement Learning for Solving the Vehicle Routing Problem" Department of Industrial and Systems Engineering Lehigh University, Bethlehem, PA 18015.