INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE
MEHTA FAMILY SCHOOL OF DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

DISSERTATION REPORT
ON

"ELECTRIC VEHICLE ROUTING PROBLEM USING DEEP REINFORCEMENT LEARNING"

Submitted by

Narendra Singodia - 21565014

Under the Guidance of

Prof. Manu Kumar Gupta

Submission Date: 02th June 2023

ACKNOWLEDGEMENT

I would like to express my deep appreciation and gratitude to my thesis supervisor, **Prof. Manu Kumar Gupta**, for his invaluable guidance, support and encouragement throughout the research process. His profound knowledge, expertise and insights have been instrumental in shaping the direction and scope of the thesis. His constructive feedback and constructive criticism have helped me improve my work and develop my skills as a researcher. I'm also grateful to my supervisor for her unwavering commitment to my academic and professional development. He has always been available to answer my questions, discuss my ideas and provide me with helpful advice, necessary resources, and facilitate communication with other experts in the field.

Overall, I'm deeply grateful to my supervisor for his exceptional support, mentorship, and guidance. I'm confident that his contributions will continue to shape my career and academic journey in the years to come. Finally, I would like to extend my gratitude to my academic institution and all the staff members who have provided me with the necessary resources, including access to the library, lab and computing facilities to complete this research. Without the support of all my fellow students in the department, this thesis would not have been possible. I'm extemely grateful to each and every one of them for their support and contributions to my academic journey.

ABSTRACT

The Vehicle Routing Problem (VRP) is a well-known combinatorial optimization problem with numerous practical applications, such as optimizing delivery routes, transportation logistics, and mobile resource allocation. Traditional methods for solving VRP often rely on heuristics and mathematical programming techniques, which may struggle to handle large-scale instances and dynamic environments. In recent years, reinforcement learning (RL) has emerged as a promising approach for addressing complex optimization problems.

In this thesis, we propose a novel framework for solving the VRP using reinforcement learning techniques. Our approach leverages the power of deep RL algorithms, specifically the combination of deep neural networks and actor critic, to learn effective policies for route planning and optimization. By formulating the VRP as a Markov Decision Process (MDP), we develop an RL agent that learns to make sequential decisions on vehicle movements and load allocations.

We evaluate our proposed RL framework on a set of benchmark VRP instances, comparing its performance against traditional heuristics and optimization techniques. The experimental results demonstrate that our approach achieves competitive solution quality and computational efficiency, especially in larger problem instances. Furthermore, we investigate the robustness and generalization capability of the learned policies by evaluating their performance on unseen problem instances.

Overall, our work highlights the potential of reinforcement learning as a promising methodology for solving the Vehicle Routing Problem. By combining the strengths of deep RL algorithms and problem-specific insights, we show that RL can offer efficient and effective solutions to this challenging optimization problem, paving the way for further advancements in the field of transportation logistics and route planning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The vehicle routing problem (VRP) is a combinatorial optimization and can be formulized as integer programming problem (ILP). In a vehicle routing problem, there is a set of depots, vehicles and customers, and the problem is to find the route for each vehicle such that each customer is serviced by any vehicle. The route is such that it can be interpreted based on your minimization criteria, e.g., traveled distance, time, or a combination of both. It first appeared in a paper by George Dantzig and John Ramser in 1959 [1], in which the first algorithmic approach was written and was applied to petrol deliveries

1.1 VRP Variant

1.1.1 Capacitated Vehicle Routing Problem

The capacitated vehicle routing problem (CVRP) extends the regular VRP by introducing a capacity element for each customer. In the literature, it is sometimes referred to as a demand. The customer's demand is $d \in \mathbb{N}^+$ which may represent capacity in the form of weight, size but also in some abstract concepts such as a basket of apples. Additionally, each vehicle has a predefined capacity $Q > 0$. If the vehicle capacity of the fleet stays the same, we are dealing with CVRP with a homogeneous fleet. A fleet with varying capacity for each vehicle is a heterogeneous fleet.

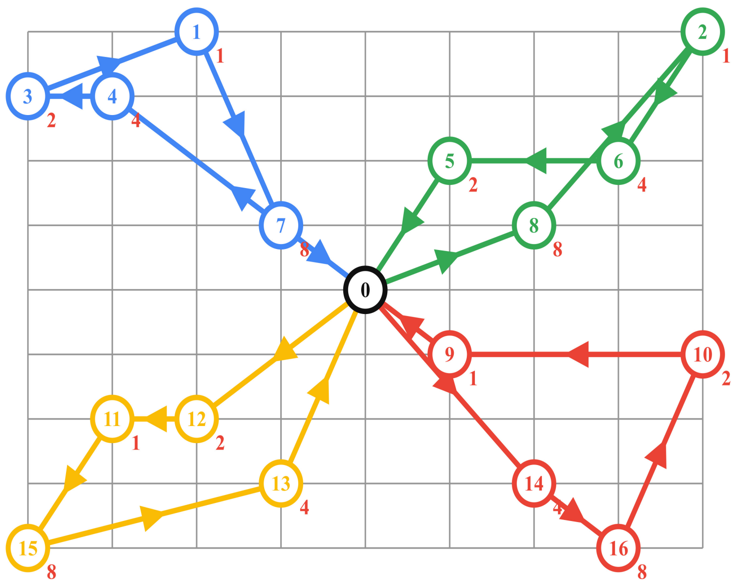


Figure 1.1: Example of CVRP instance with 4 customers and a depot. Customer are denoted by circle and depot is denoted by rectangle and demand of customer written inside in circle[3]

Mathematically, the problem can be formalized as ILP with Boolean decision variables $x_{ij}^k \in \{0,1\}$, indicating whether k vehicle travel through edge i, j to serve customer j. We also introduce state variables K_i^k to keep track of the remaining capacity of vehicle k before serving customer i.

Where c_{ij} = The costs of travelling from any customer i to any other customer j,

m= Number of vehicles,

n= Number of customers,

q_i= Demand of customer i

$$min \sum_{k=1}^{m} \sum_{i=0}^{n} \sum_{j=1,j\neq i}^{n+1} x_{ij}^k c_{ij}$$

$$s.t \sum_{k=1}^{m} \sum_{j=1,j\neq i}^{n+1} = 1 \quad \forall i \in \{1,2..,n\}$$

$$s.t \sum_{j=1,j\neq i}^{n+1} x_{ij}^k - \sum_{j=0,j\neq i}^{n} x_{ij}^k = 0 \quad \forall k \in \{1,2..,m\} \quad \forall i \in \{1,2..,n\}$$

$$\sum_{j=1}^{n+1} x_{0j}^k \leq 1$$

$$x_{ij}^k(K_i^k + q^i - K_j^k) = 0 \; \forall k \in \{1,2,...,m\} \; \forall i \in \{0,1,...,n\} \; \forall j \in \{1,2,...,n+1\} \backslash \{i\}$$

$$K_0^k = K_{max} \quad \forall k \in \{1,2,...,m\}$$

$$K_i^k \geq q^i \quad \forall k \in \{1,2,...,m\} \quad \forall i \in \{1,2..,n\}$$

$$x_{ij}^k \quad \in \{0,1\} \quad \forall k \in \{1,2,...,m\} \quad \forall i \in \{1,2,...,n\} \quad \forall j \in \{0,1,...,n\} \backslash \{i\}$$

1.1.2 Vehicle Routing Problem with Time Windows

The VRPTW extends the regular VRP by time constraint for each customer. Customers have assigned a time window interval $[ts_i, te_i]$ where $ts_i < te_i$. The time interval is the request within a vehicle that is supposed to visit the node.

The time window can be either implemented as a hard constraint or a soft constraint. Hard constraint forces the vehicle to visit the node, i.e., the customer either in the given time interval or the solution is not feasible. Soft constrains are not strictly enforcing the vehicle to visit the customer, but they introduce a penalty for a violated interval barring a penalty cost. The penalty becomes a part of the cost function which VRP aims to minimize. We will be focusing on Hard constraints for time windows.

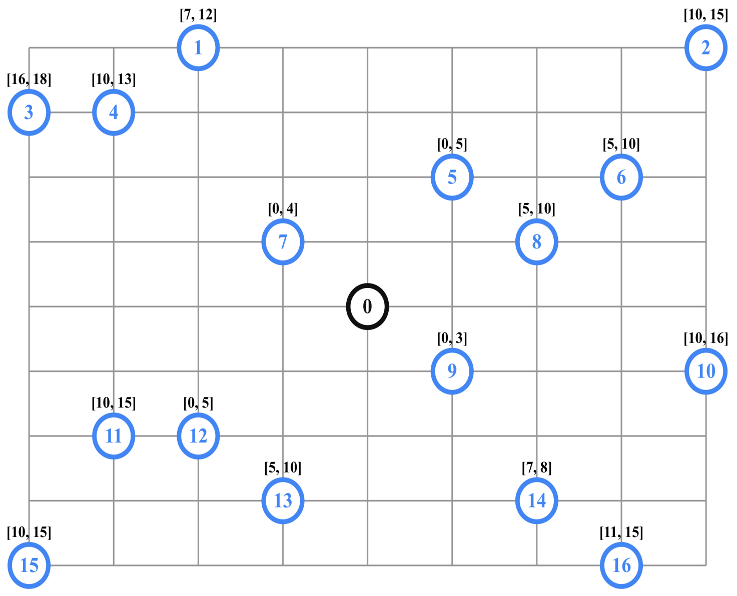


Figure 1.2: Example of VRPTW instance with 16 customers and a depot. Customer are denoted by blue circle and depot is denoted by black circle and demand of customer written inside in circle and time window [2]

1.2 VRP and other similar problem

Travelling Salesman Problem (TSP) is very well-known problem that is closely related to VRP, in this problem we have n customer and one vehicle to visit all cities such that we travel shortest path. The E-VRP, which extend the TSP by considering vehicle as Electric Vehicle which add an additional constrain of battery charge, so vehicle has to maintain positive battery charge and vehicle can go to charging state when needed. G-VRP is a special case of EVRP that doesn't consider the vehicle capacity

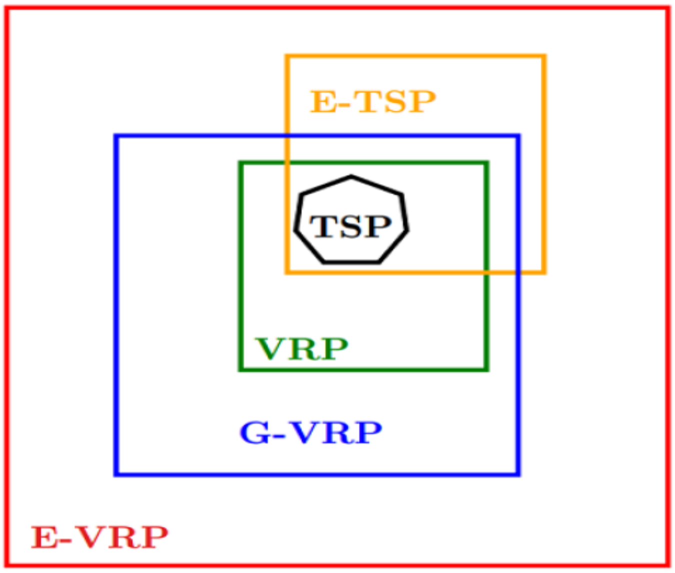


Figure 1.3: Venn diagram for several routing problems according to their complexity

1.3 Need of VRP and scale of VRP

The advent of pandemic has accelerated the process of shifting the shopping habits of consumers from traditional brick and mortar retail stores to online e-commerce websites. With these accelerating trends, one of the critical challenges faced by the e-commerce and postal industries is the transportation and delivery of orders to a large number of consumers (hundreds to potentially thousands in a day or shift). It is estimated that up to 50% of transportation costs are reserved for the last mile segment. At its core, the last mile delivery is a vehicle routing problem (VRP) which involves finding optimal routes to distribute goods from depot to customers while minimizing the total cost incurred and is well-known to be NP-hard. Adding extra constraints on vehicle capacity and time windows (TW) makes the problem even harder The scale of problem is also very high which consist of large number of customers to be serviced

Chapter 2

Literature Review

Now that we have presented and formalized the problem and a few of its variants, we will review some classical methods to solve it to optimality. Our goal in this section is not to review in detail all optimizations method, but to give an overview of the base methods involved.

2.1 Approaches to Solve the VRP

2.1.1 Integer Linear Programming

These approaches directly address the problem using the integer linear programming (ILP) formulation introduced above. The goal is to find an assignment of the binary decision variable $x_{ij}^{k} \in \{0,1\}$ minimizing the objective while satisfying constraints. Most algorithms rely on a Divide-and-Conquer approach and split the initial solution space into smaller sets. One of the basic algorithmic structures implementing this approach is called Branch-and-Bound

2.1.2 Evolutionary Algorithms

Evolutionary algorithms are heuristic method which does not give exact solution but give close to optimally solution. Evolutionary algorithms, in particular Genetic Algorithms (GAs), have been successfully used to VRP. From a population of initial candidate solutions, a new generation of solutions is generated by recombining and mutating the most promising solutions according to a fitness criterion. The process is iterated on many generations. The size of the population and the mutation encourage diversity in the exploration of solutions, while the fitness criterion guide this exploration towards the best solutions.

2.1.3 Reinforcement Learning

Reinforcement learning had phenomenal success in game playing, energy saving, bio-sciences etc. It is also used to solve Integer Programming Problem which is present in this paper ‘Reinforcement Learning for Integer Programming: Learning to Cut’ [8] and vehicle routing problem is also an integer programming problem so it can also solve using reinforcement learning. Now we will review some research paper which are try to solve VRP using reinforcement learning.

‘Deep Reinforcement Learning Algorithm for Fast Solutions to Vehicle Routing Problem with Time-Windows ‘[4] this paper solves vehicle routing problem with time window which is published by Abhinav Gupta, Supratim Ghosh, Anulekha Dhara. They used deep Q network and heuristics to solve it fast .

Deep Reinforcement Learning Approach to Solve Dynamic Vehicle Routing Problem with Stochastic Customers’ [9] this paper solves dynamic vehicle routing problem which is published by Waldy Joe, Hoong Chuin Lau. They used approximate value function and heuristics to solve it.

‘Deep Reinforcement Learning for the Electric Vehicle Routing Problem with Time Windows’ [10] this paper solves Electric vehicle routing problem which is published by Bo Lin, Bissan Ghaddar, Jatin Nathwani. They used policy gradient approach .

‘ Reinforcement Learning for Solving the Vehicle Routing Problem ’ [3] this is the first paper which tried to solve vehicle routing problems using reinforcement learning which is publish NIPS and written by Mohammadreza Nazari , Afshin Oroojlooy , Lawrence V. Snyder , Martin Takác ̌ .They tried to model vehicle routing problem as markov decision process (MDP) and have generalized their framework to include a wider range of combinatorial optimization problem such as VRP and we have also tried this approach and got some result which is given in Section 6.

Chapter 3

Theoretical Background

In this chapter, we will be covering the advanced theoretical background to fully understand the solved task of VRP using ML.

3.1 Reinforcement Learning

ML can be divided with a little simplification into three categories; supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is the most common where the model is learned from the provided labeled data. Unsupervised learning, on the other hand, is about finding a hidden patterns in a collection of data with no labels. Finally, reinforcement learning has no labeled data but learns by interacting with the environment and getting feedback in the form of rewards as shown in Figure .

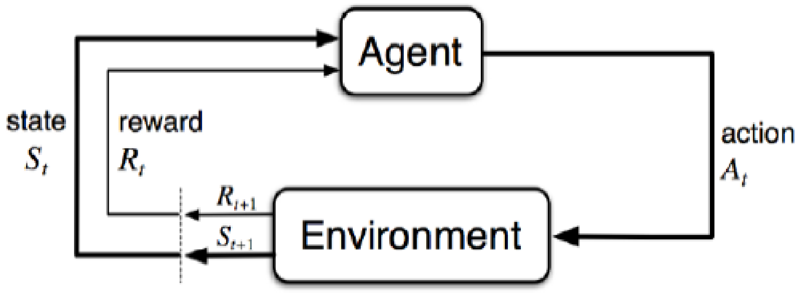


Figure 3.1: Agent feedback loop [12]

The Reinforcement Learning mimics the learning process of humans beings. By experiencing the world and accumulating knowledge, we are learning how to handle novel situations. reinforcement learning (RL) system consists of agent in observed state s_t, the agent interacts with the environment via its actions a_t at discrete time steps t and receives a reward r_{t+1} for given action. The action moves the agent into a new state s_{t+1}. The goal of the agent is to learn a policy π which chooses the action that maximizes the agent's rewards based on the environment [12].

3.1.1 State and Action Value Functions

Transition to a new state gives us a reward and to maximize it, we need a way to quantify how good a state is. A state-value function $V_\pi(s)$ predicts a future reward for a given state when following the policy π [12] .

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The equation calculates G_t, all future rewards, sometimes called as return [12]. The $\gamma \in [0,1]$ is a discount factor and penalizes the rewards in the future, incorporating the possible uncertainty and variance of the future rewards.

We will also define action-value $Q_\pi(s,a)$ which is for a similar purpose as state-value function but predicts the reward for action and state following the policy π.

$$Q_\pi(s,a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

The decomposition of state-value and action-value function replays on Bellman equations [13]. The decomposition of state-value function is

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s]$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) | S_t = s]$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$

Similarly, this method is applicable to action-value function

$$Q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s, A_t = a]$$

$$Q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{\alpha \sim \pi} Q(S_{t+1}, a) | S_t = s, A_t = a]$$

3.1.2 State and Action Value Functions

Policy Gradient [14] is a method for solving the reinforcement learning problem and learning the policy that maximizes the rewards. We define a set of parameters θ that directly models the policy, $\pi_\theta(a|s)$.

To optimize θ for the best reward, we define an objective function [14] as

$$J(\theta) = \sum_{s \in S} d_{\pi_\theta}(s) V_{\pi_\theta}(s)$$

where $d_{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ , the probability of ending in a given state [15]. The objective function $J(\theta)$ optimizes the θ parameters via gradient ascent [16].

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$$

However, computing $\nabla J(\theta_t)$ is tricky because it depends on the action selection and the stationary distribution of states [17]. Policy gradient can be simplified using Policy Gradient Theorem by Sutton et al [14].

The proof of policy gradient theorem is quite long and complicated, but you may go through it in this article [17] which is inspired by Sutton and Barto [12] . Policy gradient is simplified to the form as

$$\nabla J(\theta_t) = \mathbb{E}[\nabla \ln \pi(a|s,\theta) Q_\theta(s,a)]$$

3.1.3 REINFORCE

REINFORCE algorithm, proposed by Williams [18] in 1992, is a policy gradient method to update the policy parameter θ.

Let us define the additional terms required by the REINFORCE algorithm. We define a trajectory τ which is a sequence of states, actions, and rewards. Episode is a trajectory which ends at the terminal state S_t

$$\tau = (S_0, A_0, R_0, S_1, A_1, R_1, \ldots)$$

REINFORCE algorithm computes the policy gradient as follows

$$\nabla J(\theta) = \mathbb{E}[G_t \nabla \ln \pi(A_t|S_t,\theta)]$$

It is a simplification of a regular policy gradient because $Q_\pi(s,a) = \nabla[G_t|S_t = s, A_t = a]$ and in REINFORCE algorithm we rely on a full trajectory where we can estimate Gt based on Monte-Carlo method which is describe in this article.

Algorithm 1 REINFORCE algorithm

```
1: Result: Updated $\theta$ that maximises reward
2: Initialize $\theta$ at random
3: Generate one episode $S_0, A_0, R_0, ..., S_t$
4: for all $t = 1, 2, ..., T$ do
5:     Estimate the return $G_t$ since the time step $t$
6:     $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla \ln \pi(A_t|S_t, \theta)$
7: end for
```

3.2 Recurrent Neural Network

Recurrent Neural Network(RNN) is a kind of neural network with memory that works best with sequential data. Whenever the points in the dataset are dependent on the other points in the dataset, the data is said to be Sequential data. A conventional neural network actually makes the assumption that the data is not sequential and that each data point is independent of all the others. As a result, the inputs are examined separately, which may be problematic if the data contains dependencies. Google Voice Search and Apple's Siri which deals with sequential data employ RNN. A basic understanding of forward propagation and backward propagation is required before diving deeper into RNNs.

3.2.1 Forward and Backward Propagation

Data moves forward from the input layer through the hidden layers and onto the output layer. It could have one or more hidden layers and each node in the layers has complete connectivity. We do forward propagation to get the output of the model and check its accuracy and get the error. Training the neural network is done using the backward propagation method. Deep neural networks may be used if there are numerous hidden layers. We compute the error after forward propagation is finished. The network then receives a back-propagation of this error to update the weights.

To determine the partial derivatives of the error (loss function) with respect to the weights, we go back through the neural network. This partial derivative is now multiplied by the learning rate to calculate step size. To determine new weights, the step size is added to the initial weights. A neural network learns in this manner while being trained.

3.2.2 Basics of Recurrent Neural Network

A feed-forward neural network with internal memory is referred to as a recurrent neural network. The output of the current input depends on the previous computation, making RNNs recurrent in nature because they carry out the same function for every data input. The output is created, copied, and then sent back into the recurrent network. It takes into account both the current input and the output that it has learned from the prior input when making a decision.

RNNs can process input sequences using their internal state (memory), in contrast to feed-forward neural networks. They can therefore be used for tasks like connected, unsegmented handwriting recognition or speech recognition. All of the inputs in other neural networks operate independently of one another. In a nutshell, the present and recent past serve as RNN's two inputs. This is significant because an RNN can perform tasks that other algorithms are unable to because the data sequence contains essential information about what will happen next.

3.2.3 Different Types of RNNs

One to One

As seen in the image above, a one-to-one RNN ($T_x = T_y = 1$) is the most fundamental and traditional type of neural network, producing a single output from a single input. It also goes by the name ”Vanilla Neural Network.” It is employed to address typical machine learning issues.

One to Many

A type of RNN architecture called one to many ($T_x = 1, T_y > 1$) is used when there are multiple outputs for a single input. The creation of music would be a simple example of its application. RNN models are employed in music generation models to produce multiple musical pieces from a single musical note (single input).

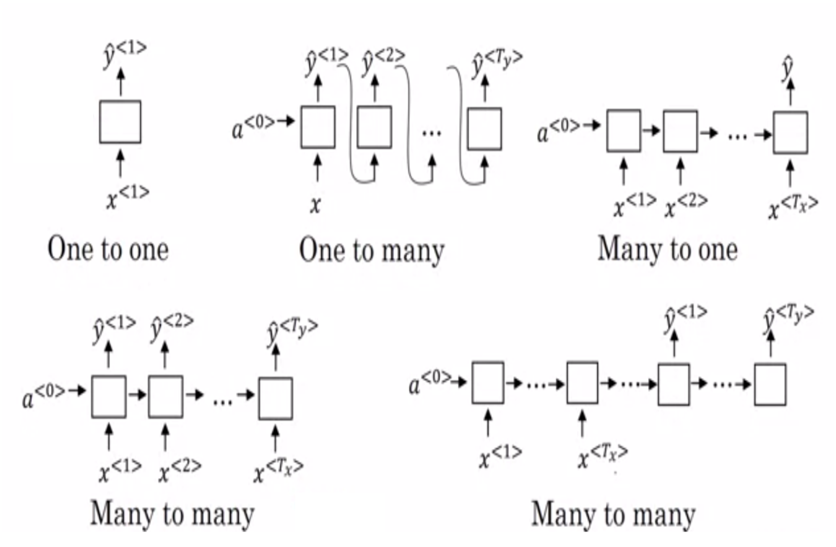
Many to One

A typical example is the many-to-one RNN architecture ($T_x > 1, T_y = 1$) used in sentiment analysis models. As the name implies, this type of model is utilized when more than one input is needed to produce a single output. For example, the sentiment analysis model for Twitter. A text input (words as multiple inputs) in that model provides its fixed sentiment (single output). Another illustration would be a system for assigning movie ratings that uses review texts as input to assign a number between 1 and 5 to a film.

Many to Many

Many-to-many RNN architecture ($T_x > 1, T_y > 1$) accepts multiple inputs and produces multiple outputs, but many-to-many models can be of two different types, as shown in the above figure:

- When input and output layers are the same sizes, this is referred to. This can also be thought of as every input having an output, and named entity recognition is a common application.
$$T_x = T_y$$
- The most prevalent use of this type of RNN architecture is in machine translation. Many-to-Many architecture can also be represented in models where the input and output layers are of different sizes. For instance, the three magical English words ”I Love You” are translated into only two in Spanish, ”te amo.” Because a non-equal Many-to-Many RNN architecture is at work in the background, machine translation models are therefore capable of returning words that are either more or less than the input string.
$$T_x! = T_y$$



$\hat{y}^{<1>}$
$x^{<1>}$
One to one
$a^{<0>}$
$\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<T_y>}$
x
One to many
$\hat{y}$
$x^{<1>}$ $x^{<2>}$ $x^{<T_x>}$
Many to one
$\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<T_y>}$
$x^{<1>}$ $x^{<2>}$ $x^{<T_x>}$
Many to many
$\hat{y}^{<1>}$ $\hat{y}^{<T_y>}$
$x^{<1>}$ $x^{<T_x>}$
Many to many

Figure 3.2: Different Types of RNNs

3.2.4 Gradient Issues in RNN Architecture

A gradient is an input-relative partial derivative. A gradient calculates how much a function's output will change if its inputs are slightly altered. A gradient is similar to a function's slope in terms of conceptualization. The steeper the slope and the higher the gradient, the more quickly a model can learn. The model halts learning if the slope is almost zero. A gradient merely quantifies the change in all weights relative to the error change. An RNN algorithm's gradient can occasionally grow too small or too large during training. As a result, in this circumstance, training an RNN algorithm becomes very challenging. This leads the poor performance, low accuracy and a prolonged training phase.

Exploding Gradient

This problem arises when we give the weights a high priority. In this situation, gradient values become excessively large, and the slope essentially grows exponentially. The following techniques can be used to resolve this:

- Identity initialization
- Truncated backward propagation
- Gradient clipping

Vanishing Gradient

This problem happens when the values of a gradient are too small, and as a result, the model stops learning or learns very slowly. The following techniques can be used to resolve this:

- Initialization of weights
- Selecting the appropriate activation function
- LSTM (Long Short-Term Memory) is the best method for resolving the vanishing gradient problem

3.2.5 Long Short Term Memory Networks(LSTM)

Sometimes, all we need to do to complete the task at hand is to look at the most recent data. Consider a language model that uses the words before it to attempt to predict the word that will come next. There is no need for additional context if we are attempting to determine the final word in ”the clouds are in the *sky*” because it is fairly clear that *sky* will come after it. RNNs can learn to use past information in such circumstances, where there is close proximity between certain information and the location where it is required.

But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. To address this problem LSTMs are introduced.

Long Short-Term Memory Networks, more commonly referred to as ”LSTMs,” are a unique class of RNN that can recognize long-term dependencies. They were first presented by Hochreiter & Schmidhuber (1997), and numerous authors refined and popularised them in subsequent works. They are now widely used and perform incredibly well when applied to a wide range of issues. Intentionally, LSTMs are created to avoid the long-term dependency issue. They don’t struggle to learn; rather, remembering information for extended periods of time is practically their default behaviour. All recurrent neural networks take the shape of a series of neural network modules that repeat.

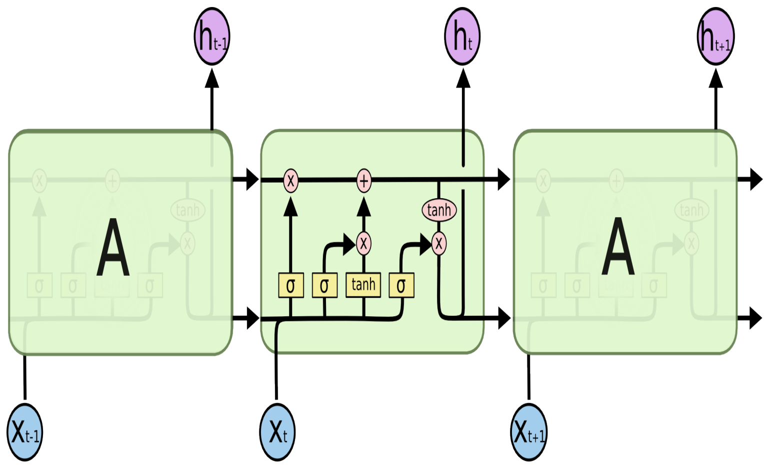


Figure 3.3: The repeating module in an LSTM Network

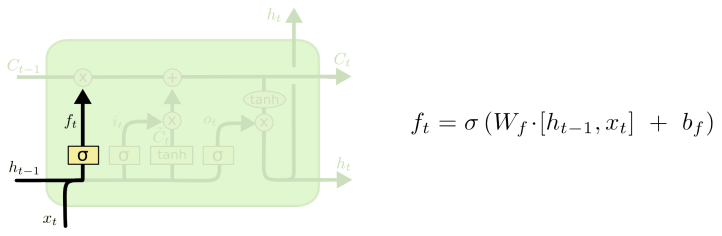
Each line in the above diagram carries an entire vector from one node’s output to another’s input. The yellow boxes are learned neural network layers, and the pink circles are pointwise operations like vector addition. Concatenation is indicated by lines merging, while lines forking indicate that their content has been copied and is being sent to various locations.

The Core Idea behind LSTM

The horizontal line that runs through the top represents the cell state is key to LSTMs. The cell state resembles a conveyor belt in some ways. With only a few minor linear interactions, it proceeds directly down the entire chain. Information can very easily continue to flow along it unchanged.The LSTM can modify the cell state by removing or adding information, which is carefully controlled by gates. Information can pass through gates on a purely optional basis. They consist of a pointwise multiplication process and a layer of sigmoid neural networks. Indicating how much of each component should be allowed through, the sigmoid layer outputs numbers between zero and one. When a value is zero, ”let nothing through,” and when a value is one, ”let everything through,” respectively. These three gates serve to safeguard and regulate the cell state in an LSTM.

Step-by-Step LSTM Walk Through

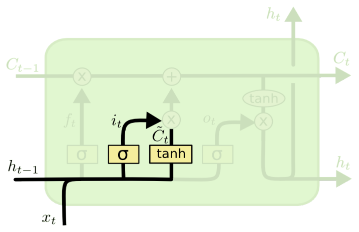
Choosing which information from the cell state to discard is the first step in our LSTM. The ”forget gate layer,” a sigmoid layer, decides on this. It examines $h_t - 1$ and x_t, and for each number in the cell state $C_t - 1$, it outputs a number between 0 and 1. A 1 means ”entirely keep this,” and a 0 means ”entirely get rid of this.”



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 3.4: The Forget Gate Layer

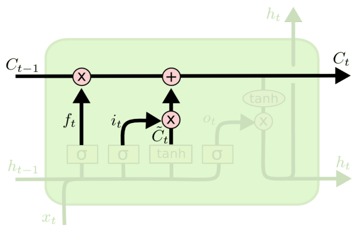
The next step is to choose the new data that will be kept in the cell state. Two parts make up this. The ”input gate layer,” a sigmoid layer, first determines which values will be updated. The state is then updated with a vector of potential new values, $\hat{C}_t$, created by a $tanh$ layer. These two will be combined in the subsequent step to produce an update to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 3.5: The Input Gate Layer

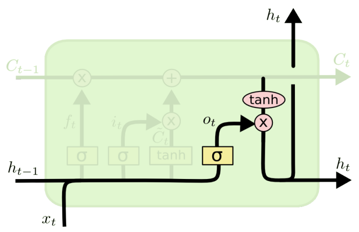
It's time to switch from the previous cell state($C_t - 1$) to the current cell state(C_t). We just need to carry out what we decided to do in the earlier steps. We multiply the previous state by f_t while omitting the earlier items on our list of things to forget. Then $i_t * \hat{C}_t$ is added. According to how much we decided to update each state value, these are the new candidate values.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.6: Updating the Old Cell State

Finally, we must choose what we will output. This output, though filtered, will be based on the state of our cell. We first run a sigmoid layer to determine which portions of the cell state will be output. Then, in order to output only the portions we decided to, we multiply the cell state by the output of the sigmoid gate after passing the cell state through tanh (to push the values to be between 1 and 1).



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3.7: The Output Gate Layer

3.2.6 Gate Recurrent Unit Networks(GRUs)

GRUs are an enhanced form of the traditional recurrent neural network. The so-called update gate and reset gate are used by GRU to address the vanishing gradient issue that plagues a standard RNN. In essence, these two vectors determine what data should be sent to the output. They have the unique ability to be trained to retain information from the past without having it fade away over time or to discard information that is unrelated to the prediction.

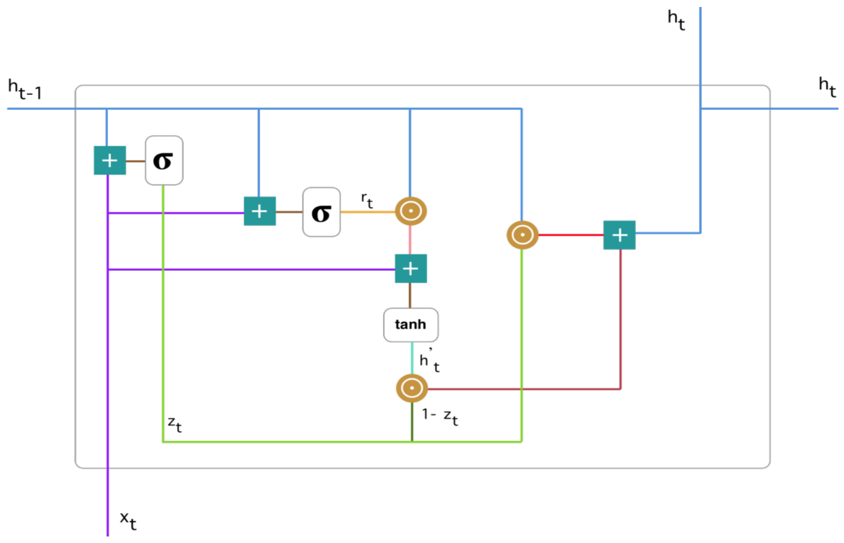


Figure 3.8: Gate Recurrent Unit

Update Gate

The update gate z_t for time step t is calculated using the formula:

$$z_t = \sigma(Wx_t + Uh_{t-1})$$

When x_t is connected to the network unit, its weight $W(z)$. The same is true for h_{t-1} which is multiplied by its own weight $U(z)$ and contains information for the previous t-1 units. Together, the two results are added, and the result is then squeezed between 0 and 1 using a sigmoid activation function. The update gate assists the model in deciding how much historical data from earlier time steps should be transmitted to the future. That is really potent because the model has the option of copying all historical data and removing the possibility of vanishing gradient issues.

Reset Gate

In essence, the model uses this gate to determine how much of the past data should be forgotten.

$$r_t = \sigma(Wx_t + Uh_{t-1})$$

This formula is the same as the one for the update gate. The difference comes in the weights and the gate's usage.

Current Memory Content

A new memory content is introduced that uses the reset gate to store the necessary historical data. The formula is as follows:

$$h_t' = tanh(Wx_t + r_t \odot Uh_{t-1})$$

- Multiply the input x_t with a weight W and h_{t-1} with a weight U.
- Calculate the Hadamard (element-wise) product between the reset gate r_t and Uh_{t-1}. Depending on that, we'll know what to take out of the earlier time steps. Consider the situation where we need to use sentiment analysis to determine a reviewer's opinion of a book. The text begins, ”This is a fantasy book which illustrates....” and ends with, ”I didn't quite enjoy the book because I think it captures too many details,” a few paragraphs later. We only need the final section of the review to gauge how satisfied readers were overall with the book. In that case, as the neural network gets closer to the text's conclusion, it will learn to assign r_t vector values that are close to 0, wiping out the earlier sentences and concentrating only on the final ones.
- Add the results of the above steps
- Apply the nonlinear activation function tanh.

Final Memory at Current Time Step

The network must calculate h_t, a vector that stores data for the current unit and transmits it throughout the network, as the final step. That requires the update gate, which is necessary. It decides what to collect from the previous steps (h_{t-1}) and what to collect from the current memory content (h_t'). The process is as follows:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$$

- The update gates z_t and h_{t-1} should be multiplied element-by-element.
- To $(1 - z_t)$ and h_t', apply element-wise multiplication.
- Add the outcomes of the above steps.

Let's use the book review as an example. This time, the text's opening paragraph contains the most pertinent information. The model can be trained to maintain the majority of the prior knowledge while setting the vector z_t close to 1. The last section of the review, which explains the book plot, will be ignored because it is irrelevant to our prediction because $1 - z_t$ will be close to 0 because z_t will be close to 1 at this time step. We can now see how GRUs use their update and reset gates to store and filter the information. As a result, the vanishing gradient problem is resolved because the model does not continually wash out new input, but rather retains it and passes it on to the network's subsequent time steps. They can perform incredibly well even in challenging situations if properly trained.

3.3 Attention

The attention mechanism is a concept used in machine learning and neural networks, specifically in the field of natural language processing (NLP). It refers to a mechanism that allows a model to focus on specific parts of the input sequence when generating or processing the output.

In NLP tasks such as machine translation, the input and output sequences can have varying lengths, making it challenging for traditional models to effectively capture long-range dependencies. The attention mechanism addresses this issue by enabling the model to assign different weights or attention scores to different parts of the input sequence.

In simple terms, the attention mechanism allows the model to ”pay attention” to relevant information in the input sequence while generating the output. It achieves this by computing a set of attention scores that reflect the importance or relevance of each element in the input sequence to each element in the output sequence. These attention scores determine how much attention or focus the model should give to each input element when generating the corresponding output element.

The attention mechanism provides a way for the model to selectively attend to the most relevant parts of the input sequence, making it more effective at capturing dependencies and improving the quality of generated outputs. It has been widely used in various NLP tasks, including machine translation, text summarization, question answering, and more.

There are several different types of attention mechanisms commonly used in machine learning and natural language processing. Here are some of the most common types:

3.3.1 Soft Attention

Soft attention, also known as additive attention, is a type of attention mechanism where the attention scores are computed by taking the dot product between a query vector and a set of key vectors. The dot product is then passed through a softmax function to obtain normalized attention weights. Soft attention allows the model to distribute attention across all elements of the input sequence.

3.3.2 Hard Attention

Hard attention, also known as deterministic attention or content-based attention, is a type of attention mechanism that selects a single element from the input sequence based on the attention scores. Unlike soft attention, which computes a weighted sum of all input elements, hard attention directly chooses a single element to attend to. Hard attention can be more computationally efficient but requires a discrete selection process, which makes it non-differentiable and less suitable for end-to-end training.

3.3.3 Scaled Dot-Product Attention

Scaled dot-product attention is a variant of soft attention commonly used in transformer models. It computes attention scores by taking the dot product between a query vector and key vectors, similar to soft attention. However, in scaled dot-product attention, the dot products are scaled by the square root of the dimensionality of the query vector, which helps stabilize the gradients during training.

3.3.4 Self-Attention

Self-attention, also known as intra-attention or intra-modality attention, is a type of attention mechanism that operates within a single sequence. It allows the model to attend to different positions within the sequence when computing the representation of each element. Self-attention is a fundamental component of transformer models and has been highly successful in various NLP tasks.

3.3.5 Multi-Head Attention

Multi-head attention is an extension of self-attention that introduces multiple sets of queries, keys, and values. It computes attention scores and produces output representations in parallel across different ”attention heads.” Each attention head captures different dependencies and information from the input sequence, allowing the model to learn more diverse and nuanced relationships.

Chapter 4

Problem Formulation

The Vehicle Routing Problem (VRP) can be formulated as a Markov Decision Process (MDP) to be solved using reinforcement learning (RL) techniques. In this formulation, we consider a fleet of vehicles and a delivery locations that need to be serviced.

1. State Space
    - Vehicle locations: The current positions of vehicles.
    - Remaining deliveries: The set of delivery locations that are yet to be serviced
2. Action Space
    - Vehicle movements: Actions representing the movement of a vehicle to a neighboring location.
3. Rewards
    - Immediate rewards :Negative of the vehicle distance travelled
4. Transition Dynamics
    - Vehicle movements: The transition from one state to another occurs when a vehicle moves from its current location to a neighboring location.
    - Load allocations: The transition occurs when a delivery location is assigned to a vehicle, resulting in an updated state with a reduced set of remaining deliveries.
5. Termination
    - The episode terminates when all deliveries have been completed.

6. Objective
    - The goal is to learn an optimal policy that maximizes the overall rewards accumulated during the delivery process while satisfying all constraints.

By formulating the VRP as an RL problem, we aim to train an RL agent to learn a policy that can make effective decisions regarding vehicle movements and load allocations, optimizing the overall delivery process. The learned policy should adapt to different problem instances and provide efficient and high-quality solutions for the VRP.

4.1 VRP modeled as a Markov Decision Process

The problem is modeled as a MDP (Markov Decision) . The nodes correspond to customers, and the depot, which are connected through a set of paths. State space: Location of nodes, both of depot and customer nodes and associated demand and vehicle location and capacity Action space: Possible Node that we can visit in next step Reward: Negative of the total vehicle distance travelled Location of node is static element where as demand of node is dynamic element

4.2 Model Architecture

Our policy model consists of a recurrent neural network (RNN) decoder coupled with an 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 For the embedding, we use 1-dimensional convolution layers for the embedding, each customer location is also embedded into a vector of size 128 and demand is mapped to a vector in a 128-dimensional vector space and used in the attention layer. We start from an arbitrary input in X_0, where we use the pointer y_0 to refer to that input. At every decoding time $t(t = 0,1,...)$, y_{t+1} points to one of the available inputs X_t , which determines the input of the next decoder step; this process continues until a termination condition is satisfied. The termination condition is problem-specific, showing that the generated sequence satisfies the feasibility constraints. For instance, in the VRP that we consider in this work, the terminating condition is that there is no more demand to satisfy. This process will generate a sequence of length $T, Y = \{y_t, t = 0,1,...,T\}$, possibly with a different sequence length compared to the input length M (M is Number of customer). This is due to the fact that, for example, the vehicle may have to go back to the depot several times to refill. We also use the notation Y_t to denote the decoded sequence up to time t i.e., $Y_t = \{y_0, y_1, ..., y_t\}$. We are interested in finding a stochastic policy π which generates the sequence Y in a way that minimizes a loss objective while satisfying the problem constraints. The optimal policy π^* will generate the optimal solution with probability 1. Our goal is to make π as close to π^* as possible.

4.3 Input Data

In our approach, we assume that the node locations and demands are randomly generated from a fixed distribution. To be more specific, the customer and depot locations are generated randomly within the unit square [0, 1] × [0, 1]. This allows for a diverse range of spatial arrangements in the problem instances.

For simplicity and ease of explanation, we assume that the demand of each node is a discrete number, ranging from 1 to 9. These demand values are chosen uniformly at random from this discrete range. It's worth mentioning that the demand values can be generated from any distribution, including continuous ones, depending on the specific requirements and characteristics of the problem at hand.

By incorporating random generation of node locations and demands, our approach allows for the exploration of various problem instances with different spatial distributions and demand patterns. This enables us to evaluate the model's performance across a wide range of scenarios and assess its ability to handle diverse routing challenges.

4.4 Training

To train the network, we use well-known policy gradient approaches. To use these methods, we parameterize the stochastic policy π with parameters θ, where θ is vector of all trainable variables used in embedding, decoder, and attention mechanism. Policy gradient methods use an estimate of the gradient of the expected return with respect to the policy parameters to iteratively improve the policy. In principle, the policy gradient algorithm contains two networks:

- an actor network that predicts a probability distribution over the next action at any given decision step,

- a critic network that estimates the reward for any problem instance from a given state.

Let us consider a family of problems, denoted by $\mathcal{M}$, and a probability distribution over them, denoted by $\Phi_{\mathcal{M}}$. During the training, the problem instances are generated according to distribution $\Phi_{\mathcal{M}}$. We also use the same distribution in the inference to produce test examples.

We have two neural networks with weight vectors θ and ϕ associated with the actor and critic, respectively. We draw N sample problems from $\mathcal{M}$ and use Monte Carlo simulation to produce feasible sequences with respect to the current policy π_θ. We adopt the superscript n to refer to the variables of the nth instance. After termination of the decoding in all N problems, we compute the corresponding rewards as well as the policy gradient in step 14 to update the actor network. In this step, $V(X_0^n;\phi)$ is the the reward approximation for instance problem n that will be calculated from the critic network. We also update the critic network in step 15 in the direction of reducing the difference between the expected rewards with the observed ones during Monte Carlo roll-outs.

Algorithm 2 REINFORCE algorithm

```
1:  Initialize the actor network with random with random weights $\theta$ and critic network with random
    weights $\phi$
2:  **for all** iteration = 1,2,... **do**
3:      Reset gradients : $d\theta \leftarrow 0$, $d\phi \leftarrow 0$
4:      Sample $N$ instances according to $\Phi_{\mathcal{M}}$
5:      **for all** $n = 1,2,...,N$ **do**
6:          Initialize step counter $t \leftarrow 0$
7:          **repeat**
8:              choose $y_{t+1}^n$ according to the distribution $P(y_{t+1}^n|Y_t^n,X_t^n)$
9:              observe new state $X_{t+1}^n$
10:             $t \leftarrow t+1$
11:         **until** termination condition is satisfied
12:         compute reward $R^n = R(Y^n,X_0^n)$
13:     **end for**
14:     $d\theta \leftarrow \frac{1}{N}\sum_{n=1}^{N}(R^n - V(X_0^n;\phi))\nabla_\theta \log P(Y^n|X_0^n)$
15:     $d\phi \leftarrow \frac{1}{N}\nabla_\phi(R^n - V(X_0^n;\phi))^2$
16:     update $\theta$ using $d\theta$ and $\phi$ using $d\phi$
17: **end for**
```

4.5 Implementation Details

In our approach, we incorporate a 1-dimensional convolutional layer for embedding purposes. This layer takes the input sequence length as its width, the number of filters as D, and the number of in-channels as the number of elements in the input x. We have observed that training the model without an embedding layer consistently results in inferior solutions.

One possible explanation for this observation is that the policy network is capable of extracting valuable features from the high-dimensional input representations more effectively when an embedding layer is utilized. The embedding layer serves as an affine transformation, allowing the model to learn and capture essential information from the input sequence. It is important to note that the embedding layer does not necessarily preserve the exact proportional distances between the embedded inputs, in comparison to the original 2-dimensional Euclidean distances.

By employing the embedding layer, we provide the model with a more efficient and effective representation of the input sequence. This enables the policy network to better exploit the relevant features and dependencies in the data, ultimately leading to improved performance and superior solutions.

In our decoder architecture, we incorporate a single layer of LSTM RNN with a state size of 128. This LSTM layer processes the customer locations, which are embedded into a shared vector of size 128. Similarly, we employ embeddings for the dynamic elements, such as the demand (d_t^i) and the remaining vehicle load after visiting node i $(l_t - d_t^i)$. These dynamic elements are also mapped to 128-dimensional vectors and used in the attention layer.

In the critic network, we first utilize the output probabilities from the actor network to compute a weighted sum of the embedded inputs. This weighted sum is then passed through two hidden layers: a dense layer with ReLU activation and another linear layer with a single output.

To initialize the variables in both the actor and critic networks, we use Xavier initialization. During training, we employ the REINFORCE Algorithm and the Adam optimizer with a learning rate of 10^{-4}. The batch size (N) is set to 128, and we clip the gradients when their norm exceeds 2. Additionally, we apply dropout with a probability of 0.1 specifically in the decoder LSTM.

These design choices and training configurations contribute to the overall performance of our model in effectively learning and generating optimal routes for the VRP.

4.6 Route Generation with an example

We will explain the route generation with the help of an example. In shown below Figure 5.1 we have customers as a circle and depot as a rectangle with their location and demand and vehicle has max capacity limit of 10 units.

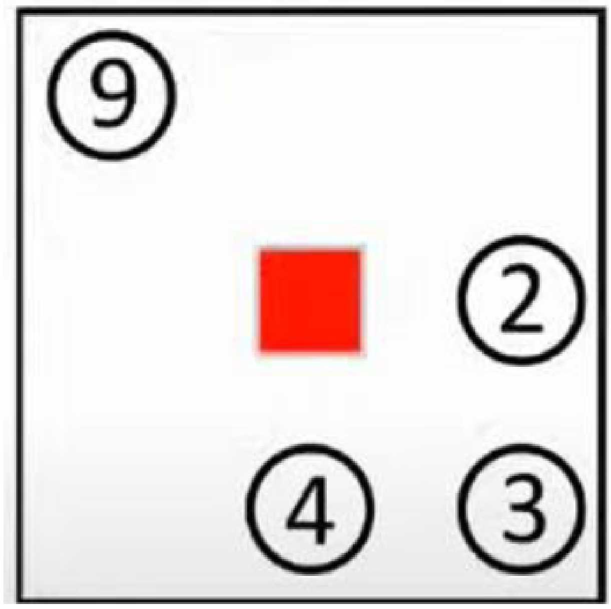


Figure 4.1: Example of CVRP instance with 4 customers and a depot. Customer are denoted by circle and depot is denoted by rectangle and demand of customer written inside in circle

State space: Location of nodes, both of depot and customer nodes and associated demand Action space: Possible Node that we can visit in next step. So, in this case is possible action are 5(depot, customer with 9 demands, customer with 4 demands, customer with 3 demands, customer with 2 demand) Reward: Negative of the total vehicle distance travelled At time step $t = 0$ all vehicle is at depot so input to RNN is depot location, feasible action is 4 (customer with 9 demand, customer with 4 demand, customer with 3 demand, customer with 2 demand) and RNN input will depot and using attention mechanism we will generate probabilities for action which will explained later.

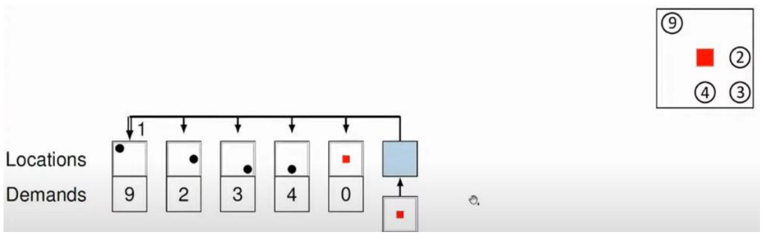


Figure 4.2: Proposed model at time step $t = 0$

Let us assume that max probability is for customer with demand 9, so we send vehicle at location of customer, so at location of this customer we are time step $t = 1$ so for this case we feed location of vehicle to RNN and generate the probability as shown below.

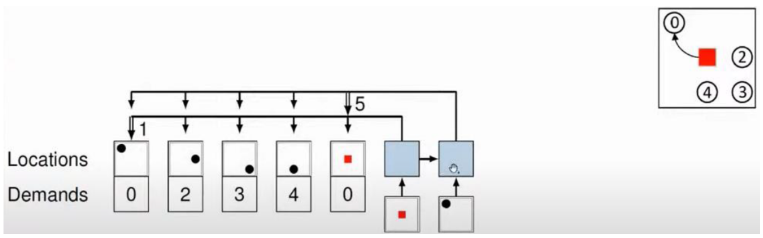


Figure 4.3: Proposed model at time step $t = 1$

Let us assume that max probability is for depot, so we send vehicle at location of depot to refill, and we are time step $t = 2$ so for this case we feed location of vehicle to RNN and generate the probability and then again select the customer which has maximum probability and we will till all feasible customer are not serviced and generate the route which shown below.

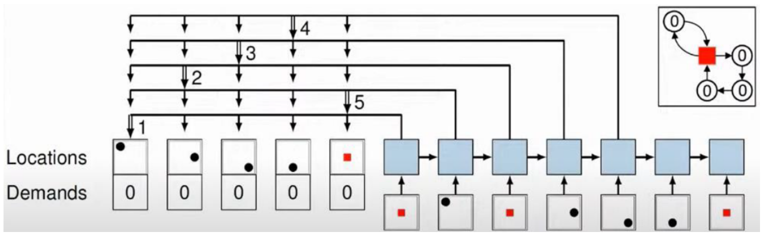


Figure 4.4: Proposed model at time step $t = T$

Now we discuss how we generate the probability using attention and RNN. We first generate embedding of location of each node and demand then we input static embedding to RNN which generate h_t and we state embedding and dynamic embedding to attention and calculate at with help of h_t and the compute context vector and the using context vector we generate probabilities for actions.

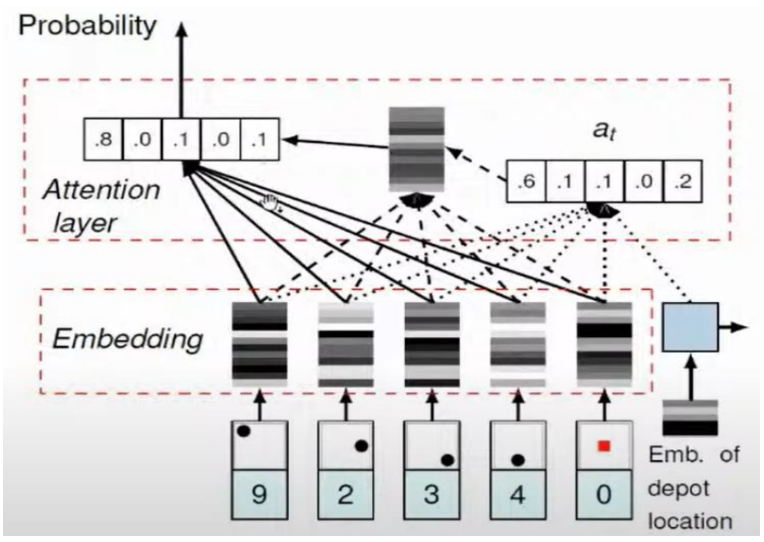


Figure 4.5: Proposed model. The embedding layer maps the inputs to a high-dimensional vector space. On the right, an RNN decoder stores the information of the decoded sequence. Then, the RNN hidden state and embedded input produce a probability distribution over the next input using the attention mechanism

Chapter 5

Results

This chapter presents the outcomes of our experiments focused on solving the Capacitated Vehicle Routing Problem (CVRP) through the application of Deep Reinforcement Learning. We conducted extensive tests on problem instances featuring 10, 20, and 50 customer nodes, each associated with different vehicle capacities of 20, 30, and 40, respectively. For instance, VRP10 comprises 10 customers with a vehicle capacity of 20. In order to tackle these diverse problem sets, we trained individual models for each specific scenario.

5.1 One Example of each model

In this study, we conducted individual model training for each problem using a randomly generated data set consisting of 100,000 instances for a duration of 10 epochs. Following the training process, we proceeded to generate new test instances. By leveraging the trained models, we obtained results that were then compared using OR Tool—an industry-standard solution provided by Google.

In order to compare the results between our model and OR Tools, we utilized the Python programming language and executed both sets of code on the same machine. However, we encountered a few discrepancies in the implementation of OR Tools for the VRP. Firstly, OR Tools only accepts integer locations for the customers and depot, whereas our problem was defined on the unit square. To address this, we resolved the issue by scaling up the problem. We multiplied all locations by 10^4, resulting in a redefined problem space of $[0,10^4] \times [0,10^4]$. After solving the problem, we scaled down the solutions and tours to obtain results for the original problem.

The second difference arises from the fact that OR Tools is designed for a VRP with multiple vehicles, each capable of having at most one tour. To accommodate this requirement, we provided a large number of vehicles to OR Tools to allow it to solve the problem freely. In contrast, our RL model does not require specifying the number of vehicles.

The third difference is related to the time limit for OR Tools to solve the VRP. We set either a specific time limit or the number of solutions that OR Tools can produce. For this study, we set the number of solutions to 650, ensuring sufficient exploration of the solution space within the given constraints.

5.1.1 For VRP 10

In our study, we trained a model specifically for instances of the VRP10 problem, which consists of 10 customers and a vehicle capacity of 20. To evaluate the performance of the model, we randomly generated test instances. Figure 5.1 provides a visual representation of one such test instance, where the red dots represent the customer locations, the black star represents the depot location, and the paths connecting them are generated using our RL model.The routes generated by our RL model resulted in a total distance traveled by the vehicles of 3.50 meters.

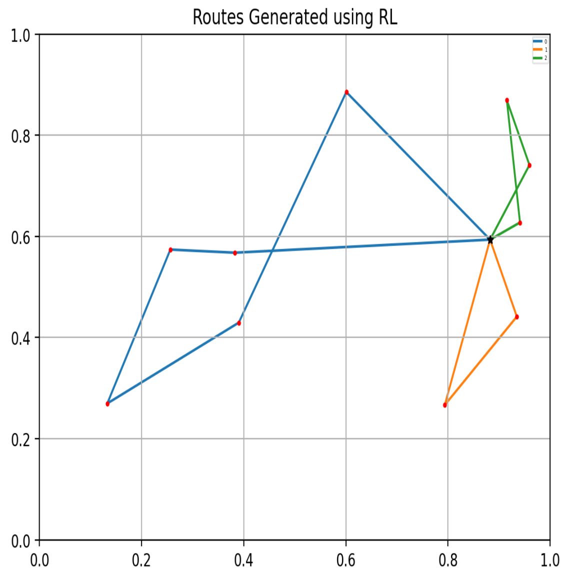


Figure 5.1: Routes Generated using RL VRP10

To compare the results with OR Tools, we utilized the same test instance and passed it to OR Tools to generate the routes. Figure 5.2 illustrates the routes generated by OR Tools for the given test instance. The total distance traveled by the OR Tools method is calculated to be 3.38 meters.

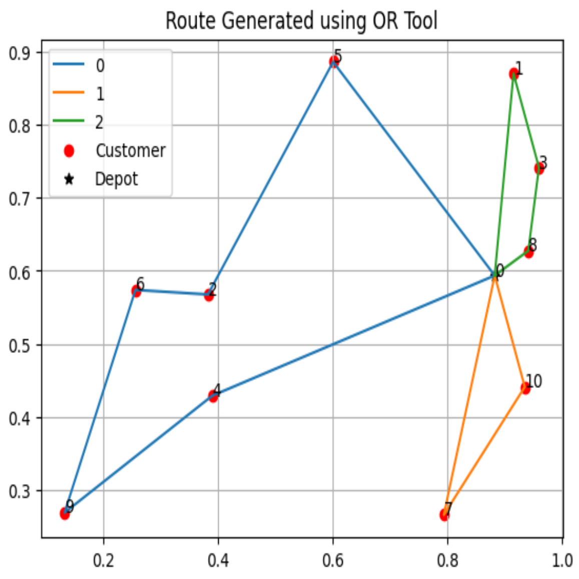


Figure 5.2: Routes Generated using OR Tools VRP10

5.1.2 For VRP20

In our study, we trained a model specifically for instances of the VRP10 problem, which consists of 20 customers and a vehicle capacity of 30. To evaluate the performance of the model, we randomly generated test instances. Figure 5.3 provides a visual representation of one such test instance, where the red dots represent the customer locations, the black star represents the depot location, and the paths connecting them are generated using our RL model.The routes generated by our RL model resulted in a total distance traveled by the vehicles of 5.00 meters.

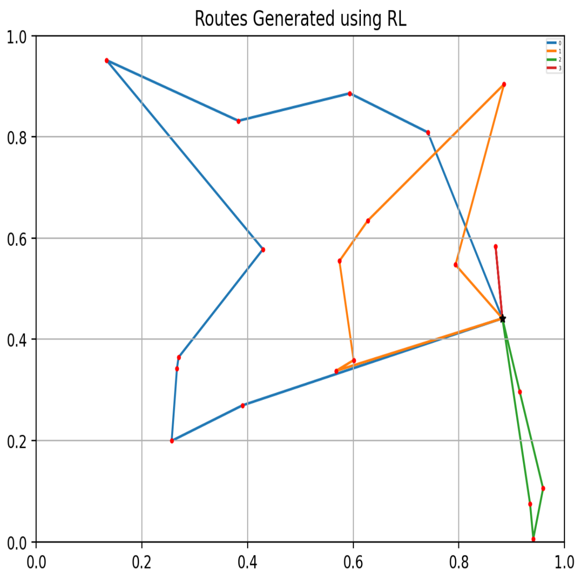


Figure 5.3: Routes Generated using RL VRP20

To compare the results with OR Tools, we utilized the same test instance and passed it to OR Tools to generate the routes. Figure 5.4 illustrates the routes generated by OR Tools for the given test instance. The total distance traveled by the OR Tools method is calculated to be 5.34 meters.

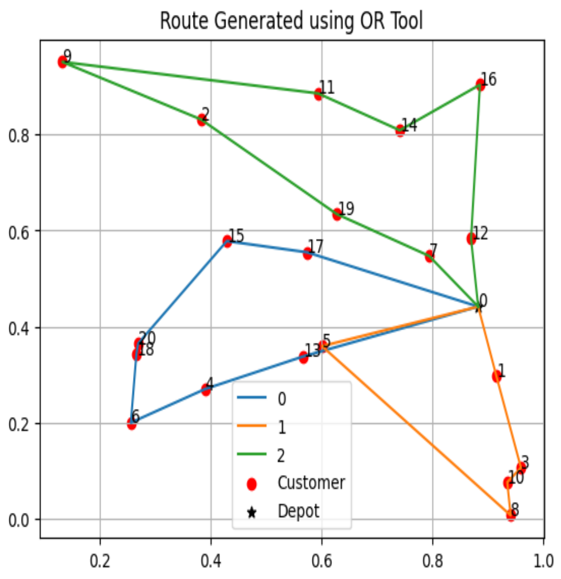


Figure 5.4: Routes Generated using OR Tools VRP20

5.1.3 For VRP50

In our study, we trained a model specifically for instances of the VRP50 problem, which consists of 50 customers and a vehicle capacity of 40. To evaluate the performance of the model, we randomly generated test instances. Figure 5.5 provides a visual representation of one such test instance, where the red dots represent the customer locations, the black star represents the depot location, and the paths connecting them are generated using our RL model.The routes generated by our RL model resulted in a total distance traveled by the vehicles of 10.01 meters.

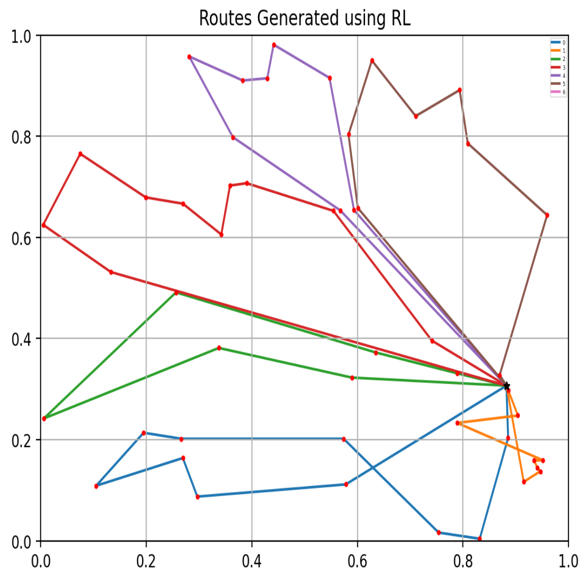


Figure 5.5: Routes Generated using RL VRP50

To compare the results with OR Tools, we utilized the same test instance and passed it to OR Tools to generate the routes. Figure 5.7 illustrates the routes generated by OR Tools for the given test instance. The total distance traveled by the OR Tools method is calculated to be 10.44 meters.

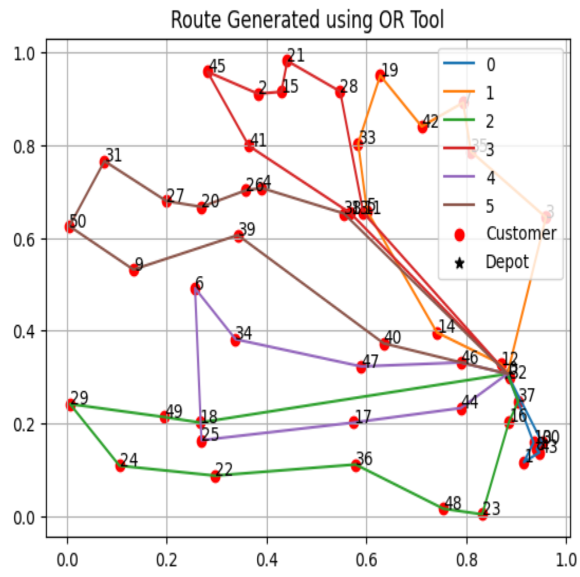


Figure 5.6: Routes Generated using OR Tools VRP50

As observed, the routes generated by both the RL model and OR Tools exhibit similarities, indicating comparable performance. Additionally, the number of vehicles used by both approaches is nearly identical. Notably, the total distance traveled by the OR Tools method is slightly lower than that of the RL model, although the difference is not significant. These findings suggest that both methods are effective in solving the problem, with OR Tools demonstrating a slightly more efficient route configuration in this particular instance.

5.2 Average comparison

Now that we have trained models for VRP10, VRP20, and VRP50, we can proceed with comparing the results between the RL model and OR Tools on average. We generated 1000 instances for each problem and obtained the routes using both methods. We then calculated the average distance traveled by the vehicles, the standard deviation of the distances, and the time taken to obtain the routes for each instance. The results are summarized in Table 5.1.

	Baseline	Mean	Std	Time(sec)
VRP10, Cap20	RL	5.84	1.03	0.006
VRP10, Cap20	OR Tool	5.51	1.32	0.901
VRP20, Cap30	RL	7.06	0.96	0.007
VRP20, Cap30	OR Tool	6.83	1.16	1.886
VRP50, Cap40	RL	12.37	1.38	0.016
VRP50, Cap40	OR Tool	11.64	1.61	7.506

Table 5.1: Average tour length, standard deviations of the tours and the average solution time (in seconds) using different baselines over a test set of size 1000

Upon reviewing Table 5.1, it becomes apparent that the average distance traveled by OR Tools is slightly less than that of the RL model. However, the difference between the two methods is not substantial. In contrast, the average time taken to solve a single problem is significantly lower for the RL model compared to OR Tools. This discrepancy suggests that RL models exhibit a notable advantage in terms of computational efficiency when solving VRP instances.

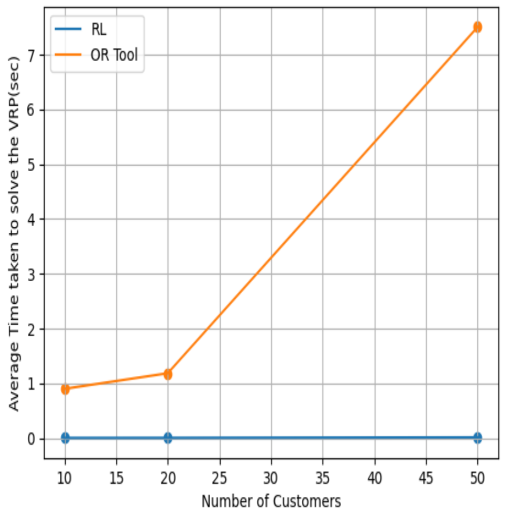


Figure 5.7: Average time taken to solve VRP

Additionally, Figure 5.7 further supports this observation. As the number of customers increases, the average time taken to solve the VRP for OR Tools exhibits an exponential growth pattern. On the other hand, the RL model's average time taken does not display such exponential behavior. This finding underscores the advantage of RL models in tackling VRP instances, as they are capable of maintaining more consistent computational performance even as problem complexity increases.

5.3 Comparison on standard data set

Up until now, our testing involved randomly generated test instances. However, we will now evaluate the performance of the VRP model on standard instances provided by Augerat. These instances consist of varying numbers of customers, and the customer locations are within a unit square box. Numerous standard methods have been applied to solve these instances, resulting in the identification of the best-known solutions to date.

In this next phase, we will employ the trained VRP50 model to solve VRPs with up to 50 customers. This means that we will apply the VRP50 model to solve VRPs with the same number of customers or less, utilizing its capabilities to handle larger problem sizes.To transform smaller VRPs into VRP50 instances, we can add additional ”fake” customers with zero demand. These fake customers serve the purpose of expanding the problem size to match the VRP50 format. By introducing these zero-demand customers, we ensure that the number of customers in the smaller

VRPs aligns with the VRP50 structure, allowing us to apply the trained VRP50 model consistently across all instances. Here are result that we get for solving standard instance in Table 5.2

Instances	Best Solution	Distance for RL(m)	Distance for OR Tool (m)	time for RL (sec)	time for OR (sec)	RL Gap %	OR Tool Gap %	Multiplier	n
B-n31-k5.vrp	672	747.3	676.1	0.259	11.969	11.2	0.608	46.28	31
A-n32-k5.vrp	784	1101.7	787.1	2.921	13.58	40.53	0.393	4.65	32
A-n33-k5.vrp	661	760.9	671.6	0.232	14.902	15.12	1.601	64.11	33
A-n33-k6.vrp	742	951.3	743.4	0.286	15.151	28.2	0.194	52.97	33
A-n34-k5.vrp	778	918.3	788.8	0.21	11.47	18.04	1.388	54.58	34
B-n34-k5.vrp	788	894.4	792.5	0.256	13.739	13.5	0.565	53.57	34
A-n36-k5.vrp	799	1051.4	810.4	0.307	14.305	31.58	1.423	46.64	36
A-n37-k5.vrp	669	859.2	672.5	0.263	14.232	28.42	0.523	54.05	37
A-n37-k6.vrp	949	1156.3	976.4	0.276	15.4	21.84	2.887	55.81	37
A-n38-k5.vrp	730	961.3	762.8	0.367	12.834	31.68	4.487	34.97	38
A-n39-k6.vrp	831	1002.4	847.3	0.316	13.806	20.63	1.965	43.69	39
A-n39-k5.vrp	822	1334.1	836.1	0.321	12.168	62.29	1.721	37.96	39
B-n43-k6.vrp	742	975.4	762.4	0.342	11.727	31.45	2.749	34.24	43
A-n44-k6.vrp	937	1139.6	987.8	0.292	15.248	21.62	5.418	52.23	44
A-n45-k6.vrp	944	1085.4	961.8	0.247	12.624	14.98	1.883	51.1	45
A-n45-k7.vrp	1146	1302.9	1162.1	0.293	13.971	13.69	1.402	47.7	45
B-n45-k5.vrp	751	1060.9	765.3	0.295	10.549	41.27	1.899	35.75	45
A-n46-k7.vrp	914	1160.7	933.3	0.371	12.452	26.99	2.111	33.52	46
A-n48-k7.vrp	1073	1296.5	1102.5	0.274	11.239	20.83	2.749	40.95	48

Table 5.2: Result summary for solving standard instance using RL and OR Tools

Table 5.2 presents a summary of the results. In this table, the ”instances” column refers to unique instance IDs provided by the author. The ”best solution” column displays the currently known best solution for each instance. The ”gap” column represents the difference between the best solution and the solution obtained by the method being evaluated. The ”multiplier” column indicates how many times faster the RL method is compared to the OR Tool approach in solving each instance. Lastly, the column labeled ”n” signifies the number of customers in each instance.

After examining Table 5.2, it is evident that the gap between the RL method and the best-known solution is larger compared to the gap between the OR Tool and the best-known solution. However, it is crucial to consider that the RL method achieves significantly faster solution times compared to the OR Tool. It is important to note that the results in Table 5.2 are obtained using the VRP50 model, which implies that the RL method is solving instances with a larger number of customers.

Taking all these factors into account, the RL method demonstrates a trade-off between solution quality (as reflected by the gap) and computational efficiency (as indicated by the significantly lower solution times). These findings emphasize the need to consider the specific requirements and constraints of the problem at hand when evaluating the performance of different approaches.

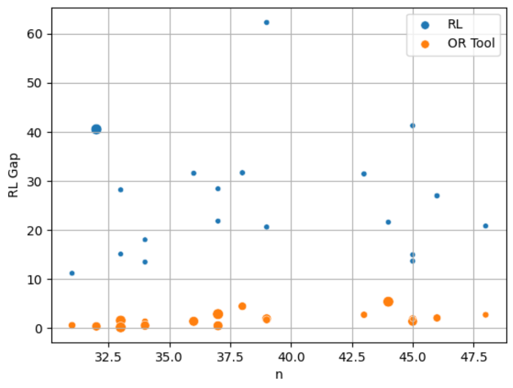


Figure 5.8: Variation between and number of customer and size of circle shows time taken solve the VRP

As the number of customers increases, the gap between the RL method and the best-known solution tends to decrease. Additionally, when the problem instances align closely with the VRP50 format on which our model is trained, the gap becomes smaller. This observation suggests that the RL method becomes more accurate and effective as it encounters problem instances that closely resemble those seen during training.

Furthermore, Figure 5.8 highlights an interesting finding: the RL method outperforms the OR Tool in terms of solution time. This demonstrates the computational efficiency of the RL approach when compared to traditional optimization methods like OR Tools.

An intriguing aspect to consider is the potential application of training different sizes of VRP models. During inference, if we encounter a problem with a different number of customers, we can utilize clustering algorithms to approximate the nearest smaller or larger VRP model for solving the given instance. This approach allows for more flexibility and adaptability when applying trained models to real-world scenarios with varying problem sizes.

5.4 Comparison on Real data Set

Now, we will proceed to test our model using real-world data and evaluate its performance. The dataset we are utilizing has been provided by Prof. Manu Kumar Gupta and contains information from a company involved in delivering goods to customers. A sample of this dataset can be seen in Table 5.3. By testing our model on real data, we aim to assess its effectiveness and applicability in practical delivery scenarios. This will provide valuable insights into the model's performance and its potential for real-world deployment.

hub name	customer id	ofd date	buyer lat	buyer long	Buyer pincode	weight(Kg)	rts wt(Kg)	transaction time(mins)	delivery slot start	delivery slot end	hub lat	hub long
Central Delhi	c-110005-0	2021-11-17	28.65	77.20	110005	375.26	0	10	4 PM	7 PM	28.657	77.21
Central Delhi	c-110007-1	2021-11-17	28.67	77.20	110007	4.81	0	5	7 AM	10 AM	28.65	77.21
Central Delhi	c-110002-2	2021-11-17	28.64	77.26	110002	248.46	0	10	4 PM	7 PM	28.65	77.21
Central Delhi	c-110008-3	2021-11-17	28.67	77.156	110008	196.53	0	10	10 AM	1 PM	28.65	77.21
Central Delhi	c-110009-4	2021-11-17	28.70	77.17	110009	63.66	0	7	4 PM	7 PM	28.65	77.21

Table 5.3: Sample Table.

In Table 5.3, we can observe various fields including the buyer's latitude and longitude, the weight of the product, the time slot for delivery, and the hub's latitude and longitude. In this dataset, there are a total of 50 customers. Since we have trained a model for the Capacitated Vehicle Routing Problem, we will focus on utilizing the buyer's latitude, longitude, product weight, and the hub's latitude and longitude. The hub will act as the depot location for our routing problem.

To input these latitude and longitude coordinates into our model, we need to convert them into x, y coordinates. For this purpose, we have employed the utm Python library. This library facilitates the conversion of latitude and longitude coordinates into x, y coordinates. By converting the coordinates, we can represent the locations in a standardized format suitable for inputting into our model. Importantly, the utm library ensures that the distances between all points are accurately maintained, especially if all coordinates belong to the same area.

Converting the latitude and longitude coordinates to x, y coordinates allows us to effectively utilize the trained model for solving the routing problem in this real-world dataset. After converting the latitude and longitude coordinates into x, y coordinates, the next step is to normalize these coordinates to fit within a unit square, which can be used as input for the model. To achieve this, we follow a two-step process.

Firstly, we subtract the minimum x-coordinate and minimum y-coordinate from all the coordinates. This ensures that all points are shifted to lie after the origin (0, 0).

Secondly, we divide all the points by the maximum distance between any two points in the dataset. By doing this, we guarantee that all the points now fall within the unit square (0, 0) to (1, 1). Importantly, this normalization step preserves the relative distances between the points.

By performing these operations, we obtain the input coordinates in a suitable format for the model. We can then apply the trained model to find the routes for this problem. The resulting routes can be visualized in Figure 5.9, providing a clear representation of the optimized paths for the delivery process.The total distance travelled by the vehicle is 7.463 m and time taken to find these path is 3.296 sec

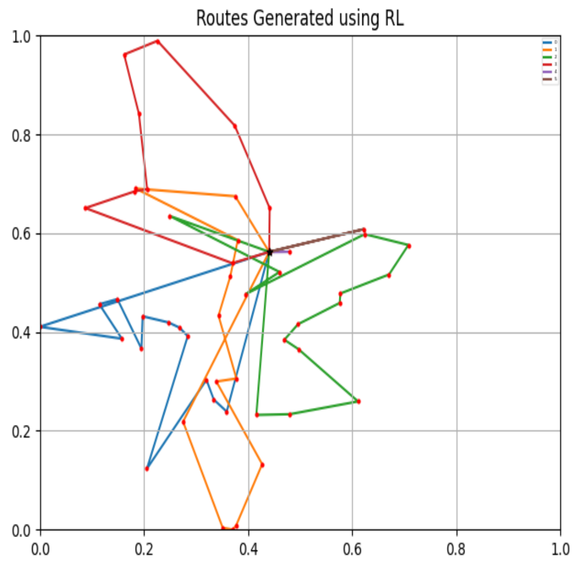


Figure 5.9: Route Generated using RL for real dataset

By providing the same input to OR Tools for solving the routing problem, we obtain the resulting routes, which can be visualized in Figure 5.10. These routes illustrate the optimized paths determined by the OR Tools approach for the given problem.

In terms of performance metrics, the total distance traveled by the vehicles using these routes is calculated to be 4.775 meters. Additionally, the time taken to find these paths using OR Tools is reported to be 14.475 seconds.

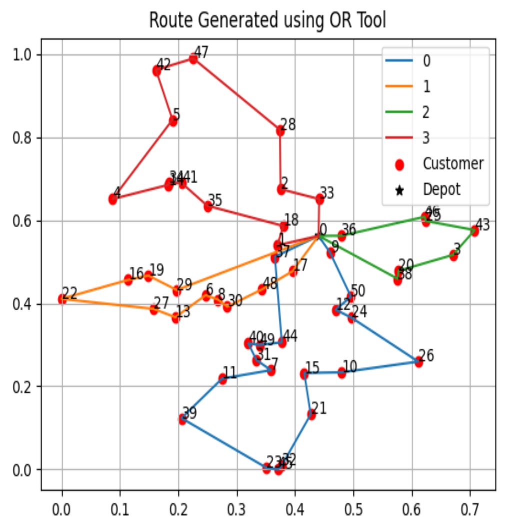


Figure 5.10: Route Generated using OR Tool for real dataset

Upon observation, it is evident that the distance traveled by the vehicles using the OR Tools approach is less compared to the RL model. This suggests that OR Tools may produce more optimized routes in terms of distance efficiency.

However, it is important to note that the RL model still performs reasonably well in terms of distance traveled. The difference in distances between the two approaches may not be significant, indicating that the RL model is capable of generating competitive routes.

One crucial aspect to consider is the time taken to find these routes. In this regard, the RL model demonstrates a notable advantage over OR Tools, as it significantly reduces the computational time required to obtain the solutions. This time efficiency is a promising characteristic of RL-based approaches, as it allows for faster decision-making and route optimization.

References

[1] *Dantzig, George B., and John H. Ramser. "The truck dispatching problem." Management science 6.1 (1959): 80-91.*

[2] *Google AI. Google's Operations Research Tools. online. 2018. url: https://developers. google.com/optimization/.*

[3] *Nazari, Mohammadreza, et al. "Reinforcement learning for solving the vehicle routing problem." Advances in neural information processing systems 31 (2018).*

[4] *Gupta, Abhinav, Supratim Ghosh, and Anulekha Dhara. "Deep Reinforcement Learning Algorithm for Fast Solutions to Vehicle Routing Problem with Time-Windows." 5th Joint International Conference on Data Science Management of Data (9th ACM IKDD CODS and 27th COMAD). 2022*

[5] *Marius M. Solomon. 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. Operations Research 35, 2 (1987), 254–265*

[6] *Guillaume Bono, Jilles S. Dibangoye, Olivier Simonin, Laëtitia Matignon, and Florian Pereyron. 2020. Solving Multi-Agent Routing Problems Using Deep Attention Mechanisms. IEEE Transactions on Intelligent Transportation Systems (2020), 1–10. https://doi.org/10.1109/TITS.2020.3009289*

[7] *Nazneen N. Sultana, Vinita Baniwal, Ansuma Basumatary, Piyush Mittal, Supratim Ghosh, and Harshad Khadilkar. 2021. Fast Approximate Solutions using Reinforcement Learning for Dynamic Capacitated Vehicle Routing with Time Windows. ArXiv abs/2102.12088 (2021)*

[8] *Tang, Yunhao, Shipra Agrawal, and Yuri Faenza. "Reinforcement learning for integer programming: Learning to cut." International Conference on Machine Learning. PMLR, 2020.*

[9] *Joe, Waldy, and Hoong Chuin Lau. ”Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers.” Proceedings of the international Conference on Automated Planning and Scheduling. Vol. 30. 2020.*

[10] *Lin, Bo, Bissan Ghaddar, and Jatin Nathwani. ”Deep reinforcement learning for the electric vehicle routing problem with time windows.” IEEE Transactions on Intelligent Transportation Systems (2021).*

[11] *https://github.com/narendrasingodia1998/VRP*

[12] *Sutton, R. S.; Barto, A. G. Reinforcement Learning: An Introduction. Cambridge, MA, USA: A Bradford Book, 2018, ISBN 0262039249*

[13] *Bellman, R. E. The Theory of Dynamic Programming. Santa Monica,CA: RAND Corporation, 1954*

[14] *Thomas, P. S.; Brunskill, E. Policy Gradient Methods for Reinforcement Learning with Function Approximation and Action-Dependent Baselines. 2017, 1706.06643*

[15] *j2kun. Markov Chain Monte Carlo Without all the Bullshit. Sep 2016.Available from: https://jeremykun.com/2015/04/06/markov-chainmonte-carlo-without-all-the-bullshit/*

[16] *Ng, A. CS229 Lecture notes - Supervised learning, 2012.*

[17] *Weng, L. Policy Gradient Algorithms. lilianweng.github.io/lil-log, 2018. Available from: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html*

[18] *Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning, volume 8, 1992: pp. 229–256*

[19] *Bahdanau, D.; Cho, K.; et al. Neural Machine Translation by Jointly Learning to Align and Translate. 2016, 1409.0473*