Reinforcement Learning for Vehicle Routing

by Amit Sowani and Kiran Avadootha

# Introduction

The Vehicle Routing problem (VRP) is a problem faced by package delivery companies daily, when they deliver goods to many different customers. The deliveries are achieved by dispatching a fleet of vehicles from a centralized storage warehouse. The goal of the delivery company is to design a route for each vehicle so that all the customers are served by exactly one vehicle and the transportation cost of the vehicles is minimized. Additional problem complexity comes from the fact that the vehicles have a fixed storage capacity and the customers have different demands.

|  |  |
| --- | --- |
|  |  |
| VRP Input Data | Feasible Solution using 2 vehicles |

Fig 1: VRP Example

Fig 1 illustrates a small VRP and a feasible solution to that problem. The customers are labeled from 0 to 4, with 0 being the warehouse. The solution uses two vehicles which are indicated by the green and blue routes.

Formally the problem can be defined by the following parameters:

* List of N locations
  + 0 is the warehouse location where all vehicles will start and end their routes.
  + Other N-1 locations are the customer locations.
* Each location is characterized by three values.
  + x, y co-ordinates
  + Demand, d
* Number of vehicles in the fleet, V.
* Fleet vehicle capacity, c.

The cost for transportation between two locations will be assumed to be proportional to the Euclidean distance between the locations.

Our target is to find the optimal solution which minimizes the transportation cost while meeting the demand at all customer locations.

# Formulating as a Markov Decision Process (MDP)

## State Definition

The vehicle delivering the goods can be considered as the agent and everything external to it is the environment, which consists of the customer locations and even the goods currently available in the vehicle.

The state is defined in terms of the following parameters:

* The outstanding demand in each of the locations.
* The current location of the vehicle.
* The current capacity of the vehicle.

## Action Definition

The action space available to the agent is the movement from the current location to another location.

## Reward Definition

Rewards should be designed to encourage the agent to complete all the customer deliveries while minimizing the total distance travelled.

Rewards are defined based on the following parameters:

* Positive reward proportional to the fulfillment of demand at a customer location
* Negative reward proportional to the distance travelled to reach the location.
* Negative reward proportional to the outstanding demand at the end of the episode.

# System Model

The system model will provide feedback to the agent from the environment regarding the next state and the reward given an action in the current state. The task is episodic and will terminate after all the vehicles in the fleet have completed the delivery of the goods. Initially the location of the vehicle will be at the warehouse and the capacity will be set to the maximum capacity of the vehicle. All customer locations will be initialized to the initial demand.

The definition of the next state has three components of which the location is built within the action, which is defined as the movement to a specific location. If the current capacity of the vehicle is more than the current demand, then the vehicle will deliver the goods to the location. The capacity of the vehicle will be reduced accordingly and the outstanding demand at the location will be set to zero.

If the vehicle successfully delivers the goods to the location, then the agent will receive a positive reward proportional to the amount of goods delivered. Apart from the delivery component, the cost component will need to be accounted for each action. The cost will be computed as the Euclidean distance between the current location and the next location.

In addition to the state and reward definitions the model captures information about the available actions in each state. At the outset all actions (locations) are available to the agent. But as the episode progresses and the vehicle delivers goods to a customer location, that location is no longer available as an action. Additionally, the vehicle is restricted from going to a location where it is not able to satisfy the entire demand (the demand is more than its current capacity).

# Learning Algorithm

## Q-learning

The off-policy TD control algorithm Q-learning was selected to learn the optimal deterministic policy. In Q-learning, the learned action-value function, Q, directly approximates q\*, the optimal action-value function, independent of the policy being followed.

|  |
| --- |
| Q-learning Algorithm |
| Algorithm Parameters: Step Size , small ε > 0.  Initialize , for all , , arbitrarily except that  Loop for each episode:  Initialize  Loop for each step of episode:  Choose from using policy derived from  Take action , observe      Until is terminal |

### Tabular implementation on a small dataset

The algorithm was implemented using tabular state action values to test the system model. Each state is defined by location of vehicle, capacity of vehicle, number of vehicles and outstanding demand. From each state the vehicle can travel to any of the other locations. For a system with N locations, V vehicles with capacity C the size of the state action value table will be given by the following:

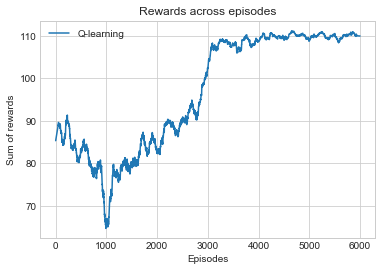
The outstanding demand is defined in a binary representation of whether the demand is fulfilled or not. This results in an exponential factor which inhibits the scaling up of this approach.

#### ε – Greedy Policy

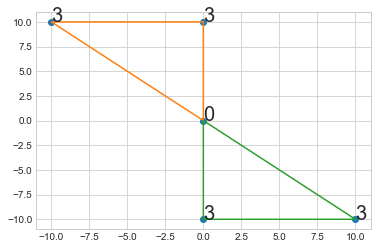
An ε – Greedy policy was selected for selecting the next action given a state. The action was selected randomly for an ε fraction of times and the rest of the time the action which resulted in the maximum state action value in the next state was selected. ε was set to a higher value initially (0.3) for more exploration and was reduced as the number of iterations increased.

#### Optimistic Initialization

Apart from the ε factor optimistic initialization was implemented to aid with early exploration of the entire state action space. The optimistic initialization results in an initial dip in the performance but once the state action values are stabilized the sum of rewards reaches close to the optimal value.



**Trajectory of sum of rewards across episodes. (α = 0.5,** ε **= 0.3)**



**Optimal Solution for the introductory problem with 4 customer locations and 2 vehicles. The values represent the demand at each location. Each vehicle has a capacity of 10.**

## Episodic Semi-gradient Q-learning

In episodic Semi-gradient Q-learning the approximate action-value function is represented as a parameterized functional form with weight vector. The approximate state-action value function is a differentiable function of the weights for all states. Training examples are selected from successive states observed in an interaction with the environment using an ε – Greedy policy. The update target will approximate the action-value function in Q-learning.

The function approximator needs to generalize across multiple state-actions while trying to minimize the error for the observed samples. The weights will be updated at each of a series of discrete time steps. Stochastic gradient-descent (SGD) adjusts the weight vector after each example by a small amount in the direction of the steepest gradient that would most reduce the error on that example.

Bootstrapping targets in the Q-learning algorithm are obtained from an estimate of the state-action function. These targets are dependent on the current value of the weight vector. Hence the targets will be biased and that they will not produce a true gradient-descent method. Bootstrapping methods take into account the effect of changing the weight vector on the estimate but ignore its effect on the target.

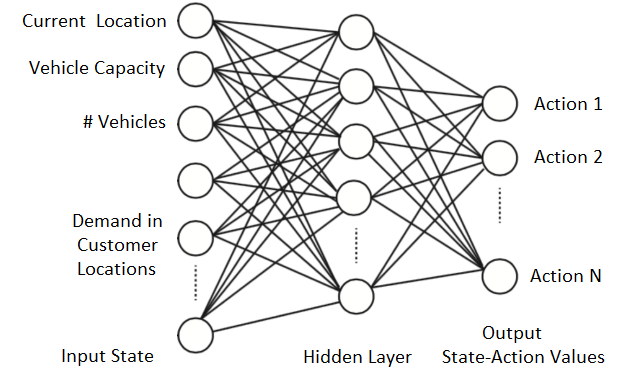
They include only a part of the gradient and, hence are called semi-gradient methods. Although semi-gradient (bootstrapping) methods do not converge as robustly as gradient methods, they do converge reliably in important cases. Moreover, they typically enable significantly faster learning and enable learning to be continual and online.

The optimal policy is identified by computing the state-action value for each possible action available in the current state and then selecting the greedy action.

|  |
| --- |
| Episodic Semi-Gradient Q-learning Algorithm |
| Input: a differentiable action-value function parametrization  Algorithm Parameters: Step Size , small ε > 0.  Initialize value function weights arbitrarily  Loop for each episode:  initial state and action of episode  Loop for each step of episode:  Choose from using policy derived from  Take action , observe  If is terminal:    Go to next episode  Else: |

### Action value estimation using a Neural Network function approximator

Neural Networks are good for nonlinear function approximation. A generic feedforward NN with one hidden layer has been used. A ReLU function is used to capture the non-linearities. The input layer represents the current state of the system which is defined by the current location of the vehicle, its capacity, the number of vehicles remaining and one node for the outstanding demand at each customer location. The output layer represents the state-action values with each node representing one of the N possible actions (locations). A real-valued weight is associated with each link.



#### Saxe Weights Initialization

Unsupervised pretraining of initial weights help in converging at the optimal solution faster. Based on the implementation by Saxe the weights are initialized as the orthogonal projection of a random matrix and a gain factor that depends on the non-linearity.

The weights and for the first and second layers respectively are initialized using the Saxe initialization.

The biases and are initialized as zeros.

#### State Action Value Computation

The neural network has one hidden layer rectifier linear units (ReLUs) which pass their input if it is bigger than one and return 0 otherwise. ReLU gates have nice properties like the sparsity of the activation and having non-vanishing gradients. The output of the neural network is the estimated state-action values for all actions for a given state. It is a linear function of the hidden units to estimate the value of a continuous target.

The state action values are computed as follows:

#### Gradient Computation

The update the weights of the neural network gradient of the state-action value function with respect to the weights for a given input are computed using backpropagation.

Where denotes the element wise multiplication and is the gradient of the ReLU activation function which is an indicator whose ith element is 1 if and 0 otherwise.

#### ADAM Optimization Algorithm

Adam (Adaptive Moment Estimation) is an optimization method in which the [learning rate](https://en.wikipedia.org/wiki/Learning_rate) is adapted for each of the parameters. The learning rate for a weight is adjusted based on a running average of the magnitudes of recent gradients and the second moments for that weight.

The running estimates of the first and second moments are computed as follows:

and are parameters of the algorithm and is the gradient times the TD error.

To get unbiased estimates of the mean and second moment Adam defines and as follows:

Weights are updated as follows:

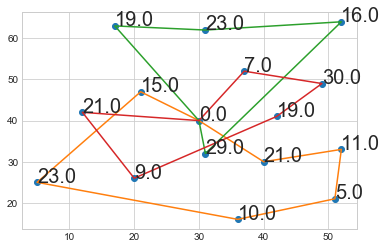
is the step size parameter and is a small parameter to avoid the denominator becoming zero.

#### Fixed Target Network Weights

A separate network was used for generating the TD target to improve the stability of the online Q-learning with neural networks. After a fixed number of iterations (M) the entire network of weights is cloned to obtain a target network. It is used generating the Q-learning targets for the following M updates. This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases the state action valus of the current state also increases the state-action value of the next state and hence also increases the target, possibly leading to oscillations or divergence of the policy. Generating the targets using an older set of parameters adds a delay between the time the weights are updated and the time the update affects the targets, making divergence or oscillations much more unlikely.



**Trajectory of sum of rewards across episodes. (α = 0.0001,** ε **= 0.35, 4 hidden layers)**



**Feasible Solution for a VRP with 16 customer locations and 3 vehicles. The values represent the demand at each location. Each vehicle has a capacity of 90.**