Project Report

|  |  |
| --- | --- |
| **Course Name (NICF)** | *Bundle Program-Artificial Intelligence* |
| Product Name (Marketing & Sales) | *Bundle Program-Artificial Intelligence* |
| **Module Name (NICF)** | **NICF-Reinforcement Learning (SF)** |
| Product Name (Marketing & Sales) | **NICF- Reinforcement Learning (SF)** |

|  |  |  |  |
| --- | --- | --- | --- |
| Student name | | Assessor name | |
| Sami Nathan Renganayagi | |  | |
| Date issued | Completion date | | Submitted on |
| 05/07/2021 | 09/07/2021 | | 10/07/2021 |
|  | |  | |
| Project title | Retention of good cab drivers  using Reinforcement learning model | | |

|  |
| --- |
| Learner declaration |
| I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.  No need sign  Student signature: Date: date only |

**Index**

1. **Project Overview**
2. **Project Technical Environment**
3. **Activity 1: Environment Summary**
4. **Activity 2: DQN Overview**
5. **Activity 3: Convergence Overview**
6. **Screenshots of each task of Activity 1 and its explanation**
7. **Screenshots of each task of Activity 2 and its explanation**

**8.Screenshots of each task of Activity 3 and its explanation**

**1.Project Overview: Describe the Project along with Project Outcomes (Explain the Project in your own words in 15 – 20 lines)**

The goal of the project is to build an RL-based algorithm which can help the cab drivers to maximise their profits by improving their decision-making process on the field.

**Problem Statement:**

The need for choosing the “Right “requests most drivers get a healthy number of ride requests from customers throughout the day. But with the recent hikes in electricity prices (all cabs are electric), many drivers complain that although their revenues are gradually increasing, their profits are almost flat. Thus, it is important that drivers choose the “Right” rides.i.e. choose the rides which are likely to maximise the total profit earned by the driver that day.

In this project, Env.py is to create the environment and RL agent that learns to choose the best request.

Env.py is a “environment class-each method (function)of the class has a specific purpose.

RL agent learns to pick the best request using DQN. The model has been trained with the hyperparameter (Epsilon (decay rate), Learning rate, discount factor etc.)

Training depends purely on the epsilon-function. If the decay fast, it will not let the model explore much and Q-values converge early but to suboptimal values. If the decay slowly, the model converges slowly.

2.Project Technical Environment: (Describe the MDP)

Taking long-term profit as the goal, the proposal of the method based on reinforcement learning to optimize taxi driving strategies for profit maximization. This optimization problem is formulated as Markov Decision method-based Process.

Decision Epochs:

The decisions are made at an hourly interval; thus, the decision epochs are discrete.

Assumptions:

1. The taxis are electric cars. It can run for 30 days non-stop, i.e., 24\*30 hrs. Then it needs to recharge itself. If the cabdriver is completing his trip at that time, he’ll finish that trip and then stop for recharging. So, the terminal state is independent of the number of rides covered in a month, it is achieved as soon as the cabdriver crosses 24\*30 hours.
2. There are only 5 locations in the city where the cab can operate.
3. All decisions are made at hourly intervals. We won’t consider minutes and seconds for this project. So, for example, the cab driver gets requests at 1:00 pm, then at 2:00 pm, then at 3:00 pm and so on. So, he can decide to pick among the requests only at these times. A request cannot come at (say) 2.30 pm.

4. The time taken to travel from one place to another is considered in integer hours (only) and is dependent on the traffic. Also, the traffic is dependent on the hour-of-the-day and the day-of-the-week.

State:

The state space is defined by the driver’s current location along with the time

components (hour-of the-day and the day-of-the-week). A state is defined by three

variables:

𝑠 = 𝑋𝑖𝑇𝑗𝐷𝑘 𝑤ℎ𝑒𝑟𝑒 𝑖 = 0 … 𝑚 − 1; 𝑗 = 0 …. 𝑡 − 1; 𝑘 = 0 … . . 𝑑 − 1

Where 𝑋𝑖 represents a driver’s current location, 𝑇𝑗 represents time component (more specifically hour of the day), 𝐷𝑘 represents the day of the week

• Number of locations: m = 5

• Number of hours: t = 24

• Number of days: d = 7

A terminal state is achieved when the cab completes his 30 days, i.e., an episode is 30 days long.

Actions

Every hour, ride requests come from customers in the form of (pick-up, drop) location.

Based on the current ‘state’, he needs to take an action that could maximise his monthly revenue. If some passenger is already on-board, then the driver won’t get any requests.

Therefore, an action is represented by the tuple (pick-up, drop) location. In a general

scenario, the number of requests the cab-driver can get at any state is not the same. We

can model the number of requests as follows:

The number of requests (possible actions) at a state is dependent on the location. Say,

at location A, you get 2 requests on average and at location B, you get 12 requests on

average. That means, when at A, the cab-driver is likely to get 2 customer requests

from anywhere to anywhere of the form (𝑝, 𝑞).

For each location, you can sample the number of requests from a Poisson distribution

using the mean λ defined for each location as below:

|  |  |
| --- | --- |
| Location | λ (of Poisson Distribution) |
| Location A | 2 |
| Location B | 12 |
| Location C | 4 |
| Location D | 7 |

The upper limit on these customers’ requests (𝑝, 𝑞) is 15.

Apart from these requests, the driver always has the option to go ‘offline’ (accept no ride). The noride action just moves the time component by 1 hour. So, you need to append (0,0) action to the customer requests.

There’ll never be requests of the sort where pickup and drop locations are the same. So, the action space A will be: (𝑚 − 1) ∗ 𝑚 + 1 for m locations. Each action will be a tuple of size 2. You can define action space as below:

• pick up and drop locations (𝑝, 𝑞) where p and q both take a value between 1 and m;

• (0, 0) tuple that represents ’no-ride’ action.

For example, if the set of all possible locations is of size 3: {A, B, C}. So, at state (A, 6:00 pm, Wednesday), his possible actions would be of the form: (𝑖,𝑗) where i and j can be any location from {A, B, C}, but i ≠ j. The following table shows all possible actions the driver can take.

|  |  |
| --- | --- |
| (A, B) | (C, A) |
| (A, C) | (C, B) |
| (B, C) | (B, A) |
| (0, 0) |  |

The average number of requests received when the driver is in location A is 2. So, the cab driver will get any two random requests from the above table, say (A, B) and (C, A). Also, he’ll always have the option of ‘no-ride’. So, his possible actions at that state would be: (𝐴, 𝐵), (𝐶, 𝐴), (0, 0)

His action should be to pick the best request among these three.

State Transition

Given the current state 𝑠 = 𝑋𝑖𝑇𝑗𝐷𝑘, the next state 𝑠’ will be as following:

𝑠 ′ = { 𝑋𝑞𝑇𝑡 ′𝐷𝑑 ′ 𝑎 = (𝑝, 𝑞) 𝑋𝑝𝑇𝑡 ′𝐷𝑑 ′ 𝑎 = (0,0) }

Where 𝑡 ′ , 𝑑 ′ represents the time and day respectively after taking an action.

You can calculate the total time taken to reach from one point to other from the Time Matrix (calculated basis the historical data) provided to you in the zip file. You don't need to learn a distribution of the time taken; all possible values of (pick up, drop, t, d) are available in the ‘TM.npy’ file, you simply need to look them up.

Time Matrix is a 4-D matrix. The 4 dimensions are as below:

• Start location

• End location

• Time-of-the-day

• Day-of-the-week

Python indices for these dimensions are as:

𝑇𝑖𝑚𝑒 − 𝑚𝑎𝑡𝑟𝑖𝑥 [𝑠𝑡𝑎𝑟𝑡 − 𝑙𝑜𝑐][𝑒𝑛𝑑 − 𝑙𝑜𝑐][ℎ𝑜𝑢𝑟 − 𝑜𝑓 − 𝑡ℎ𝑒 − 𝑑𝑎𝑦] [𝑑𝑎𝑦 − 𝑜𝑓 − 𝑡ℎ𝑒 − 𝑤𝑒𝑒𝑘]

This matrix has been calculated considering the distance between two locations and traffic conditions, which generally depends on the hour-of-the-day and the-day-of-the-week. To make the problem manageable, we divided the 24-hour frame into 4 segments:

from 12:00 am to 6:00 am, 6:00 am to 12:00 pm, 12:00 pm to 6:00 pm and 6:00 pm to 12:00 am.

You are given this time-matrix in the zip file shared. (Please run it once to understand its dimensions).

Reward

Your objective is to maximize the profit of a driver. Let 𝐶𝑓 be the amount of battery consumed per hour and 𝑅𝑘 be the revenue he obtains from the customer for every hour of the ride. The values of these parameters are defined in the skeleton code provided to you. Also, we have assumed that both the cost and the revenue are purely functions of time, i.e. for every hour of driving, the cost (of battery and other costs) and the revenue (from the customer) is the same - irrespective of the traffic conditions, speed of the car etc.

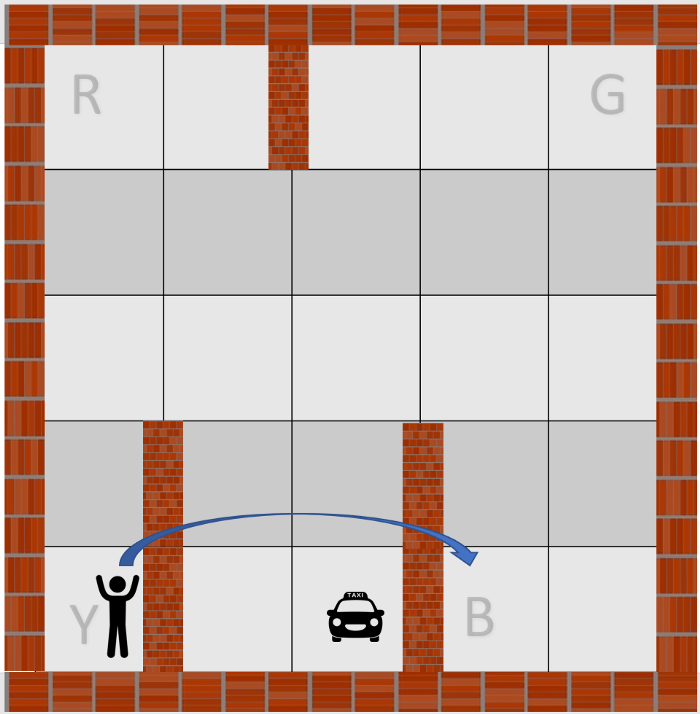
So, the reward function will be (revenue earned from pickup point 𝑝 to drop point 𝑞) - (Cost of battery used in moving from pickup point 𝑝 to drop point 𝑞) - (Cost of battery used in moving from current point 𝑖 to pick-up point 𝑝). Mathematically,

𝑅(𝑠 = 𝑋𝑖𝑇𝑗𝐷𝑘) = { 𝑅𝑘 ∗ (𝑇𝑖𝑚𝑒(𝑝, 𝑞)) − 𝐶𝑓 ∗ (𝑇𝑖𝑚𝑒(𝑝, 𝑞) + 𝑇𝑖𝑚𝑒(𝑖, 𝑝)) 𝑎 = (𝑝, 𝑞)

𝑎 = (0,0) }

Where 𝑋𝑖 represents a driver’s current location, 𝑇𝑗 represents time component (more specifically hour of the day), 𝐷𝑘 represents the day of the week, 𝑝 represents the pickup location and 𝑞 represents the drop location.

3.Activity1 : Environment Summary



Chart, icon

Description automatically generated

The hyperparameters are

m = 5 # number of cities, ranges from 1 ..... m

t = 24 # number of hours, ranges from 0 .... t-1

d = 7 # number of days, ranges from 0 ... d-1

C = 5 # Per hour fuel and other costs

R = 9 # per hour revenue from a passenger

Locations: A, B, C, D, E represented by integers 1, 2, 3, 4, 5 (start index 1)

Time of the day: 24 hours clock 00:00, 01:00, ..., 22:00, 23:00

represented by integers 0, 1, 2, 3, 4, ..., 22, 23

Day of the week: MON, TUE, WED, THU, FRI, SAT, SUN represented by integers 0, 1, 2, 3, 4, 5,6

state\_encod\_arch1(self, state):

convert the state into a vector so that it can be fed to the NN. This method converts a given state into a vector format.

Hint: The vector is of size m + t + d.

requests(self, state):

Determining the number of requests basis, the location. Use the table specified in the MDP and complete for rest of the locations

reward\_func (self, state, action, Time\_matrix):

Takes in state, action and Time-matrix and returns the reward

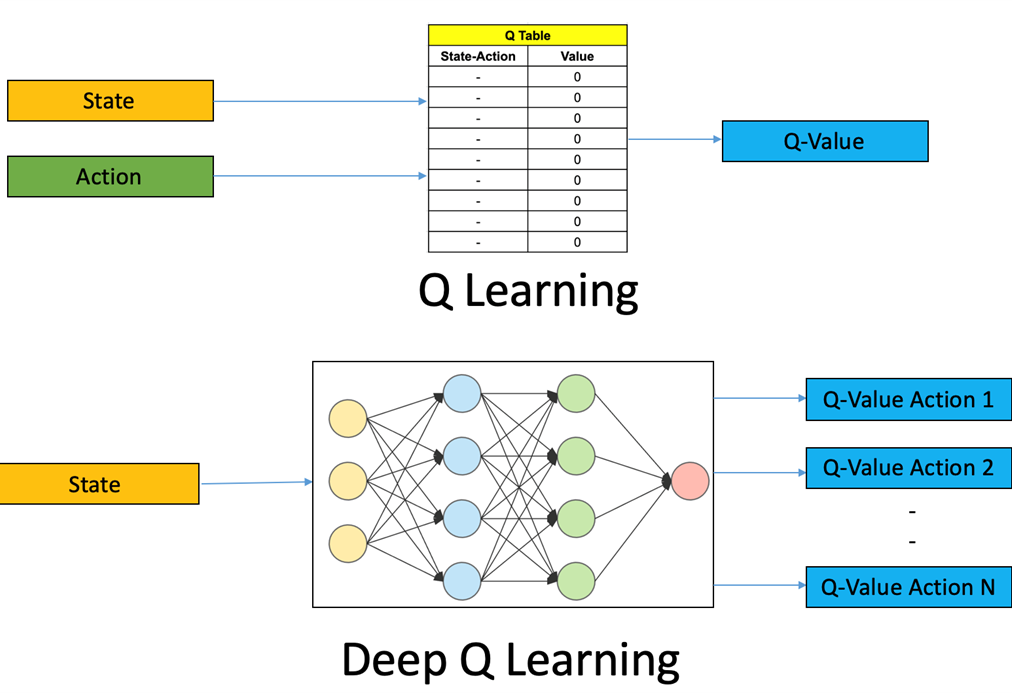
reward = (revenue earned from pickup point 𝑝 to drop point 𝑞) –

(Cost of battery used in moving from pickup point 𝑝 to drop point 𝑞) -

(Cost of battery used in moving from current point 𝑖 to pick-up point 𝑝)

4.Activity 2: DQN Overview

In [4]:

Agent Class

1. State and Action Size
2. Hyperparameters
3. Create a neural-network model in function 'build\_model()'
4. Define epsilon-greedy strategy in function 'get\_action()'
5. Complete the function 'append\_sample()'. This function appends the recent experience tuple <state, action, reward, new-state> to the memory
6. Complete the 'train\_model()' function with following logic:
   * If the memory size is greater than mini-batch size, you randomly sample experiences from memory as per the mini-batch size and do the following:
     + Initialise your input and output batch for training the model
     + Calculate the target Q value for each sample: reward + gamma\*max(Q(s'a,))
     + Get Q(s', a) values from the last trained model
     + Update the input batch as your encoded state and output batch as your Q-values
     + Then fit your DQN model using the updated input and output batch.

In [4]:

### **DQN block**

* Call the DQN agent
* While! terminal state
  + Pick epsilon-greedy action from possible Actions for the current state
  + Evaluate the reward and next state
  + Append the experience to the memory
  + Train the model by calling function agent. Train model
  + Keep track of rewards, Q values, loss

5.Activity 3: Convergence Overview

* Sample a few state-action pairs and plot their Q-values along episodes

The Q value is high around 250 episodes

* Check whether the total rewards earned per episode are showing stability

The score consistently earned in every episode .

6.Screenshots of each task of Activity 1 and its explanation

Graphical user interface, text, application

Description automatically generated

7.Screenshots of each task of Activity 2 and its explanation

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

8.Screenshots of each task of Activity 3 and its explanation

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated