Abstract

The Need for Choosing the **'Right' Requests** is very crucial since most drivers in any big town or city get a healthy number of ride requests from customers throughout the day. But with the recent hikes in fuel prices, 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 **maximize the total profit** earned by the driver that day.

For example, say a driver gets three ride requests at 5 PM. The first one is a long-distance ride guaranteeing high fare, but it will take him to a location which is unlikely to get him another ride for the next few hours. The second one ends in a better location, but it requires him to take a slight detour to pick the customer up, adding to fuel costs. Perhaps the best choice is to choose the third one, which although is medium-distance, it will likely get him another ride subsequently and avoid most of the traffic. Considering this factor our AI agent will help cab drivers pick up the best request at any time they are free.

Index

Introduction

Field of Study

Since our agent is trying to aim at maximizing profit based on the requests it picks up, so we are using Reinforced Learning (RL). Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state and creates an exact matrix for the working agent which it can “refer to” to maximize its reward in the long run. It can handle problems with stochastic transitions and rewards without requiring adaptations.

Although this approach is not wrong in itself, this is only practical for very small environments and quickly loses its feasibility when the number of states and actions in the environment increases (dynamism). The solution for the above problem comes from the realization that the values in the matrix only have relative importance i.e., the values only have importance with respect to the other values. Thus, this thinking leads us to **Deep Q-Learning** which uses a **deep neural network** to approximate the values. This approximation of values does not hurt as long as the relative importance is preserved.

The basic working step for Deep Q-Learning is that the initial state is fed into the neural network and it returns the Q-value of all possible actions as on output.

Market Feasibility Analysis

The cab services market in India was valued at INR 30.72 Bn in FY 2020 and is expected to expand at a compound annual growth rate (CAGR) of ~**12.93%** during the FY 2021 – FY 2025 period, to reach a value of **INR 55.15 Bn** by FY 2025. Statistics show that more than 91% of top surveyed companies have ongoing utilization and investment on AI (New Vantage, 2022).

Technical Feasibility Analysis

To deploy any ML model at the same time as we build a scalable framework to support future modeling activities, expect to spend closer to **$95K over the first five years** on the functionality required to deploy the model. We implemented a small scenario with only 5 locations based on our machine computational power. But in real life with increase in number of locations the complexity of state-action space. The agent can handle it but will require more computational power.

Operational Feasibility Analysis

Our model can perfectly bring out the maximum profit upon completion. This is the main requirement for any cab driving businesses.

Legal Feasibility Analysis

Our project uses information about the time taken from one location to another based on the current request time (time of the day and day of the week). This information is confidential only to the cab driving business using our agent because they will be the one providing us the data.

Area of Application

As our project depicts, the sole application of our project is cab driving businesses who are operating successfully in big metropolitan cities and helping them maximize their monthly profit by picking up the best request. However potential applications of DQN are self-driving cars, natural language processing, predicting future sales etc.

Background Study

Kun Jin, Wei Wang, Xuedong Hua, and Wei Zhou (Jiangsu Key Laboratory of Urban ITS, Southeast University, Nanjing 211189, China) worked on a similar research topic - **Reinforcement Learning for Optimizing Driving Policies on Cruising Taxis Services**, Published: 26 October 2020.

This paper developed the reinforcement learning (RL) framework to optimize driving policies on cruising taxis services. Firstly, they formulated the drivers' behaviors as the Markov decision process (MDP) progress, considering the influences after taking action in the long run. The RL framework **using dynamic programming for policy learning** and data expansion was employed to calculate the state-action value function. Following the value function, drivers can determine the best choice and then quantify the expected future reward at a particular state.

By utilizing historic orders data in Chengdu, they analyzed the function value's spatial distribution and demonstrated how the model could optimize the driving policies. Finally, the realistic simulation of the on-demand platform was built. Compared with other benchmark methods, the results verified that the new model performs better in **increasing total revenue, answer rate and decreasing waiting time**, with the relative percentages of **4.8%, 6.2% and 27.27%** at most.

***M*ethodology of this research paper**

Algorithm 1: Policy learning using dynamic programming

```
Input: Dataset that stores the historical state transitions pair: $P = \{(s_i, a_i, r_i, s\prime_i)\}$
Output: $V(s)$, $Q(s,a)$
1: Initialize $V(s)$, $Q(s,a)$, auxiliary matrix $N(s,a)$ as zeros for all state space.
2: Each state consists of a time index and a location index: $s_i = (t_i, l_i)$.
3: K is the maximum time index.
4: for $t$ = K-1 to 0 do
5:    Collect subset $P_t = \{(s_i = (t_i, l_i), a_i, r_i, s\prime_i) \mid t_i = t\}$
6:    for each $(s_i, a_i, r_i, s\prime_i)$ in $P_t$ do
7:       $N(s_i, a_i) \leftarrow N(s_i, a_i) + 1$
8:       if $a_i = 1$ then
9:          $Q(s_i, 1) \leftarrow Q(s_i, 1) + \frac{1}{N(s_i,1)}\left[\gamma^{\Delta t} V(s\prime_i) + R_\gamma - Q(s_i, 1)\right]$
10:      else
11:         $Q(s_i, 0) \leftarrow Q(s_i, 0) + \frac{1}{N(s_i,0)}\left[\gamma^{\Delta t} V(s\prime_i) - Q(s_i, 0)\right]$
12:      end if
13:   end for
14:   $V(s) = MAX(Q(s,0), Q(s,1))$
15: end for
```

Algorithm 2: Transitions expand using spatiotemporal search

```
Input: Dataset that stores the historical state transitions pair: $P = \left\{\left(s_i, a_i, r_i, s'_i\right)\right\}$
Output: Expandable transitions set $P$
1: Initialize $P_{expand} = \{\}$
2: State space $S = \{s = (t, l)\}$, parameter $t_{lim}$
3: for each $s = (t, l)$ in $S$ do
4:    Collect subset $P_s = \{(s_i, a_i, r_i, s'_i) | s_i = s\}$
5:    if $P_s = \varnothing$ then
6:       Spatial Search:
7:          Find nearby location index set $L^\sim = \left\{l^\sim \middle| Dis(l^\sim, l) = 1\right\}$,, $Dis(l^\sim, l)$ is the manhattan distance between $l^\sim$ and $l$.
8:       for $l^\sim$ in $L^\sim$ do
9:          if $P_{l^\sim} = \{(s_i, a_i, r_i, s'_i) | s_i = (t, l^\sim)\} \neq \varnothing$ then
10:            Random choose $(s_i, a_i, r_i, s'_i)$ from $P_{l^\sim}$, add $(s_{new} = (t, l), a_i, r_i, s'_i)$ to $P_{expand}$
11:            go to line 3
12:         End if
13:      End for 1
14:      Temporal Search:
15:      for $t^\sim$ in max($0, t^\sim - t_{lim}$) to min( $K - 1, t^\sim + t_{lim}$) do
16:         if $P_{t^\sim} = \{(s_i, a_i, r_i, s'_i) | s_i = (t^\sim, l)\} \neq \varnothing$ then
17:            Random choose $(s_i, a_i, r_i, s'_i)$ from $P_{t^\sim}$, add $(s_{new} = (t, l), a_i, r_i, s'_i)$ to $P_{expand}$
18:            go to line 3
19:         End if
20:      End for
21:   End if
22: End for
23: $P = P_{expand} + P$
```

Proposed Methodology

Our project is similar to this project except for the fact that we used a **Deep Q neural network** as the brain of our agent than simple Q – network. Since deep learning models have more learning capacity than other models. Thus, it can show improvement in performance even with dynamic environment with changing state and action space.

Environment Simulation

We use Markov Decision Process (MDP) to describe the environment. The Markov Decision Process (MDP) is typically used to model sequential decision-making problems, which is the necessary process of RL

State space: States we are focusing on are Locations, Time of the day and Day of the week. We send the state space as a vector to our learning model. The length of the vector will be 5+24+7=36 since there are 5 cities, 24 hours in a day and 7 days in a week.

For example, the state at city location 4, at 5th hour of Tuesday will be represented as.

[0 0 0 **1** 0 |0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |0 0 **1** 0 0 0 0]

Action Space: The action space is the possible combinations of travelling between any two locations. The total action space size is $5C_2 + 1 = 21$. Because we can get request to go from any one of 5 cities to another and adding 1 because we consider action (0,0) as cab driver rejecting the request.

Reward Function: Reward function = (revenue earned from pickup point p to drop point q) - (Cost of fuel used in moving from pickup point p to drop point q) - (Cost of fuel used in moving from current point i to pick-up point p)

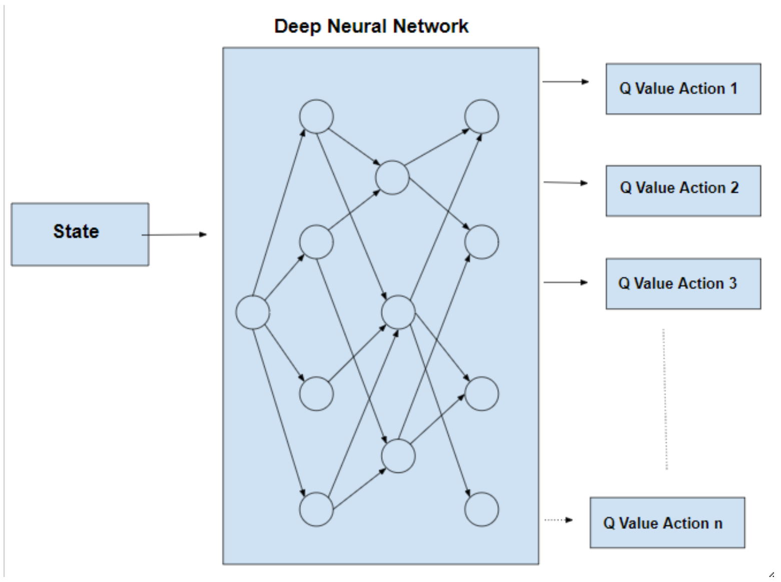
Terminal State for an episode: When the total time of driving exceeds 720 hours i.e., number of hours in a month.

Requests Generator (Probability of picking a request and move to next state): We use Poisson distribution for generating probable number of requests that can come if the cab driver is at a particular location (maximum limit set to 15). Based on the number we randomly pick up that many number of samples from action space and add it to possible action space. If we do not include (0,0) then we include that in our possible action space.

Next state transition: Once a drop is done and we need to move to the next state. The new state will be defined as conjunction of new location, new time of the day and new day of the week.

New location will be the drop location i.e., second value of the previous action chosen

Agent Design



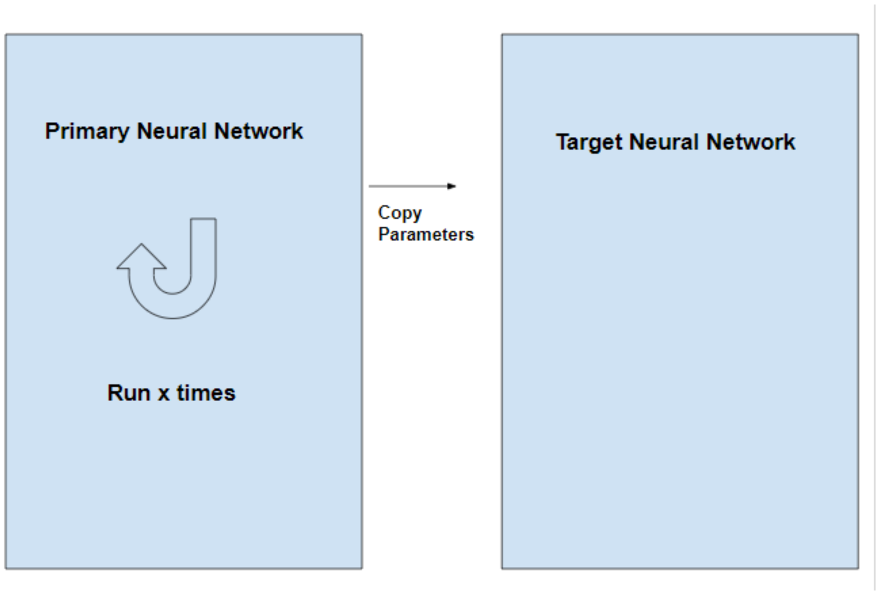
DQN Agent

Sensor: A pop up notification or sound when a request is generated
Actuator: A tick signal beside the best request on the phone screen

Algorithm

```
Initialize $Q_0(s,a)$ for all pairs (s,a)
s = initial state
k = 0
while(convergence is not achieved)
{
    simulate action a and reach state s'
    if(s' is a terminal state)
    {
        target = R(s,a,s')
    }
    else
    {
        target = R(s,a,s') + $\gamma max_{a'} Q_k(s',a')$
    }
$\theta_{k+1} = \theta_k - \alpha\Delta_\theta E_{s'\ P(s'|s,a)}[(Q_\theta(s,a) - target(s'))^2]|_{\theta=\theta_k}$
   s = s'
}
```

The equation **target = R(s,a,s') +** $\gamma max_{a'} Q_k(s',a')$, the term $\gamma max_{a'} Q_k(s',a')$ is a variable term. Therefore, in this process, the target for the neural network is variable unlike other typical Deep Learning processes where the target is stationary.
This problem is overcome by having two neural networks instead of one. One neural network is used to adjust the parameters of the network and the other is used for computing the target and which has the same architecture as the first network but has frozen parameters. After an x number of iterations in the primary network, the parameters are copied to the target network.



Agent Brain Design

γ is a discount factor, that tells how important future rewards are to the current state. Discount factor is a value between 0 and 1. A reward R that occurs N steps in the future from the current state, is multiplied by γ^N to describe its importance to the current state. For example, consider γ = 0.9 and a reward R = 10 that is 3 steps ahead of our current state. The importance of this reward to us from where we stand is equal to the value (0.9^3) *10 = 7.29.

E Epsilon refers to the probability of choosing to explore over exploiting most of the time with a small chance of exploring. It starts with 1 which means the agent doesn't know anything and it is free to choose any action. Epsilon decay is the rate at which the rate of exploration stops decreasing after each episode. Slowly when the model starts learning it will stop exploring like before. Epsilon min is the minimum epsilon value after which it should not decrease. This helps prevent overfitting.

α Learning rate is how big we take a leap in finding optimal policy. In Q Learning it's how much we are updating the Q value with each step. Higher alpha means Q values are being updated in big steps. When the agent is learning we should decay this to stabilize your model output which eventually converges to an optimal policy.

Implementation

Language Used – Python

Data used – "TM.npy"; A 4-dimensional numpy array storing time value from one city to another at a particular time of the day and day of the week
https://www.kaggle.com/datasets/maverickscientist/tm-file

Libraries and Modules used –

- ***sys*** - The sys module in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows **operating on the interpreter** as it provides access to the variables and functions that interact strongly with the interpreter.

- ***os*** - The OS module in Python provides functions for creating and removing a **directory** (folder), fetching its contents, changing and identifying the current directory, etc.

- ***numpy*** – NumPy is a python library **used for working with n dimensional arrays**. It also has functions for working in domain of linear algebra, Fourier transform, and matrices.

- ***pandas*** - Pandas is an open-source Python package that is most widely used **for data science/data analysis and machine learning tasks**. It is built on top of another package named NumPy, which provides support for multi-dimensional arrays.

- ***random*** - Random is a Python module is an in-built module of Python which is used to generate **random numbers**.

- ***time*** - The time module provides many ways of representing **time in code,** such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code.

- ***math*** - The Python Math Library provides us access to some common **math functions and constants** in Python, which we can use throughout our code for more complex mathematical computations.

- ***collections*** - Collections in Python are containers used for storing data and are commonly known as data structures, such as lists, tuples, arrays, dictionaries, etc. We are using **dequeue**.

- ***pickle*** - Pickle module is **used for serializing and de-serializing python object structures**.

- ***matplotlib*** – Matplotlib is a comprehensive library for creating static, animated, and **interactive visualizations** in Python.

- ***tensorflow*** - TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on **training and inference of deep neural networks**.

- ***keras*** - Keras is an open-source software library that provides a Python interface for **artificial neural networks**.

Results and Discussion

Agent Testing (Env.py)

Test 1

```
agent = CabDriver()
agent.test_run()
```

```
CURRENT STATE: (4, 3, 6)
REQUESTS: [(4, 1), (1, 0), (4, 3), (3, 1), (2, 0), (1, 3), (2, 4), (0, 0)]
REWARDS: [8.0, 14.0, 8.0, 6.0, -1.0, 6.0, -9.0, -5]
NEW POSSIBLE STATES: [[1, 5, 6], [0, 11, 6], [3, 5, 6], [1, 9, 6], [0, 14, 6], [3, 9, 6], [4, 12, 6], [4, 4, 6]]
MAXIMUM REWARD: 14.0
ACTION : (1, 0)
NEW STATE: [0, 11, 6]
NN INPUT LAYER: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

Test 2

```
agent = CabDriver()
agent.test_run()
```

```
CURRENT STATE: (2, 5, 4)
REQUESTS: [(3, 1), (2, 0), (1, 0), (4, 1), (2, 1), (0, 4), (4, 2), (0, 0)]
REWARDS: [-18.0, 12.0, -23.0, 19.0, 28.0, -15.0, 27.0, -5]
NEW POSSIBLE STATES: [[1, 14, 4], [0, 8, 4], [0, 15, 4], [1, 12, 4], [1, 12, 4], [4, 8, 4], [2, 14, 4], [2, 6, 4]]
MAXIMUM REWARD: 28.0
ACTION : (2, 1)
NEW STATE: [1, 12, 4]
NN INPUT LAYER: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
```

Test 3

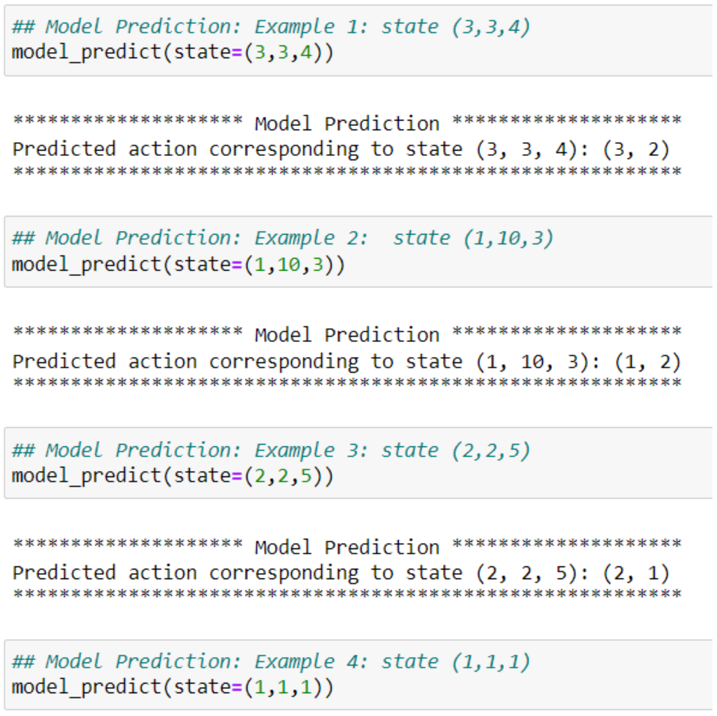
```
agent = CabDriver()
agent.test_run()
```

```
CURRENT STATE: (2, 10, 2)
REQUESTS: [(1, 3), (4, 2), (2, 1), (1, 4), (4, 0), (3, 0), (4, 1), (0, 0)]
REWARDS: [-16.0, -12.0, 16.0, -20.0, -16.0, -7.0, -20.0, -5]
NEW POSSIBLE STATES: [[3, 15, 2], [2, 16, 2], [1, 14, 2], [4, 14, 2], [0, 15, 2], [0, 15, 2], [1, 14, 2], [2, 11, 2]]
MAXIMUM REWARD: 16.0
ACTION : (2, 1)
NEW STATE: [1, 14, 2]
NN INPUT LAYER: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
```

Our agent is successfully being able to pick a state from state space and pick up the best request from the coming requests by calculating the reward from accepting each request and picking the best rewarding one. Thus, it being able to transition to its new state based on the action it takes and also encodes it to a vector to feed into our NN model.

Model Testing (Agent_Architecture.ipynb)

We trained the model up to 20000 iterations, discount factor = 0.95, learning rate = 0.01, epsilon = 1, max epsilon = 1, epsilon decay = -0.00045, min epsilon = 0.0000001, batch size = 32



```
## Model Prediction: Example 1: state (3,3,4)
model_predict(state=(3,3,4))
```

```
******************** Model Prediction ********************
Predicted action corresponding to state (3, 3, 4): (3, 2)
**********************************************************
```

```
## Model Prediction: Example 2:  state (1,10,3)
model_predict(state=(1,10,3))
```

```
******************** Model Prediction ********************
Predicted action corresponding to state (1, 10, 3): (1, 2)
**********************************************************
```

```
## Model Prediction: Example 3: state (2,2,5)
model_predict(state=(2,2,5))
```

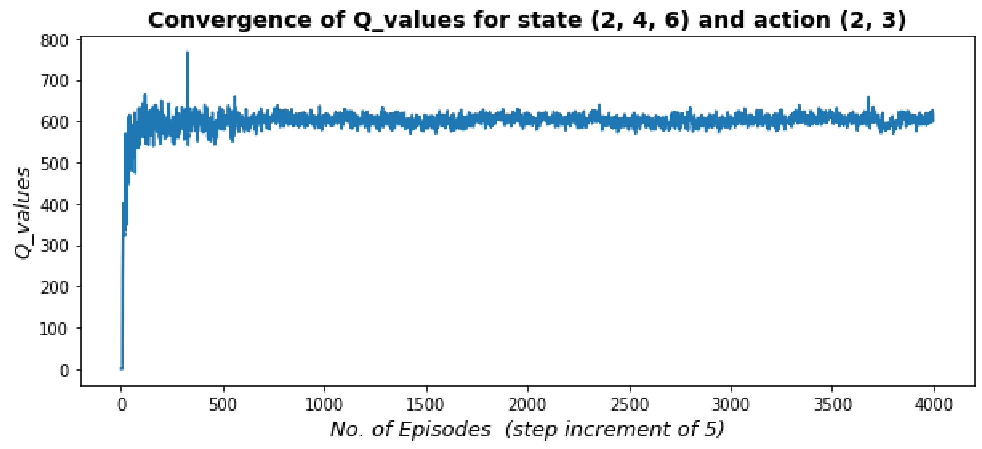
```
******************** Model Prediction ********************
Predicted action corresponding to state (2, 2, 5): (2, 1)
**********************************************************
```

```
## Model Prediction: Example 4: state (1,1,1)
model_predict(state=(1,1,1))
```

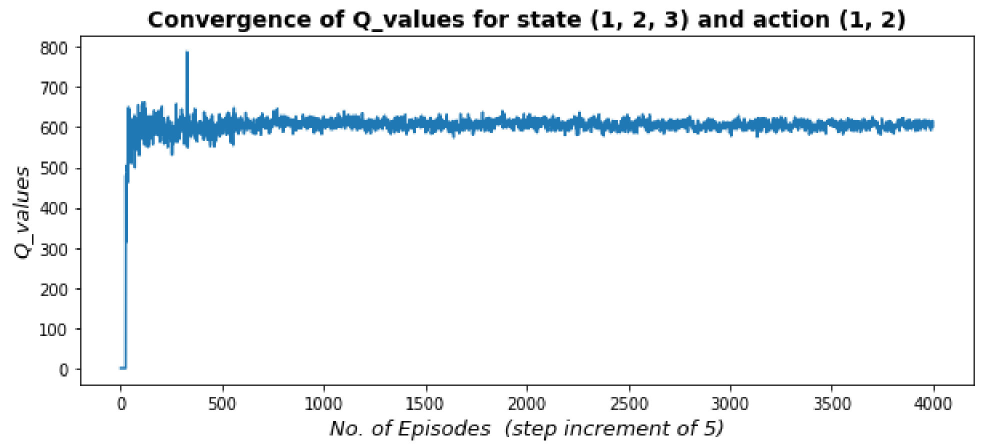
The neural network model successfully predicts the best action (request) according to what it learnt given any input state.

Checking for Convergence

```
Tracking Convergence for state-action pair 1: State (2,4,6), Action (2,3)
```

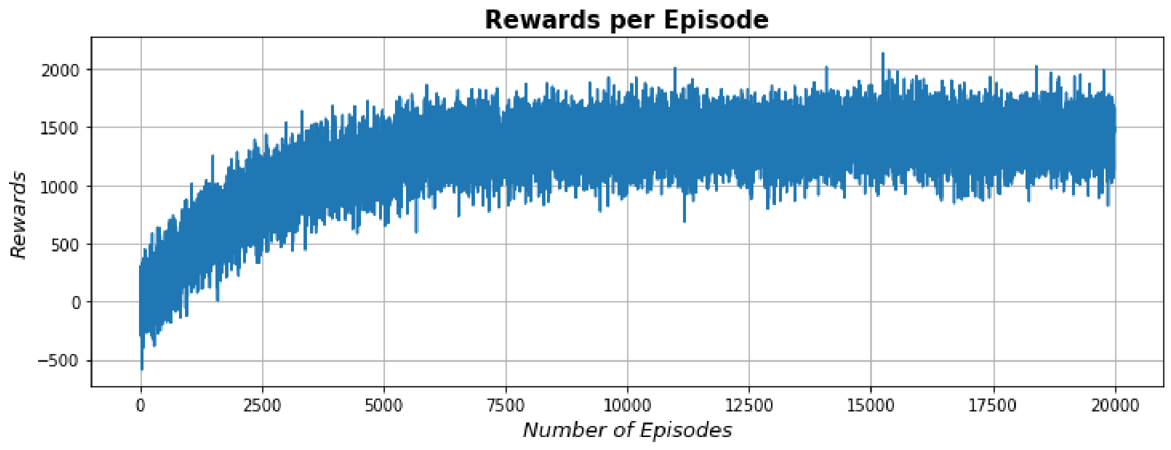


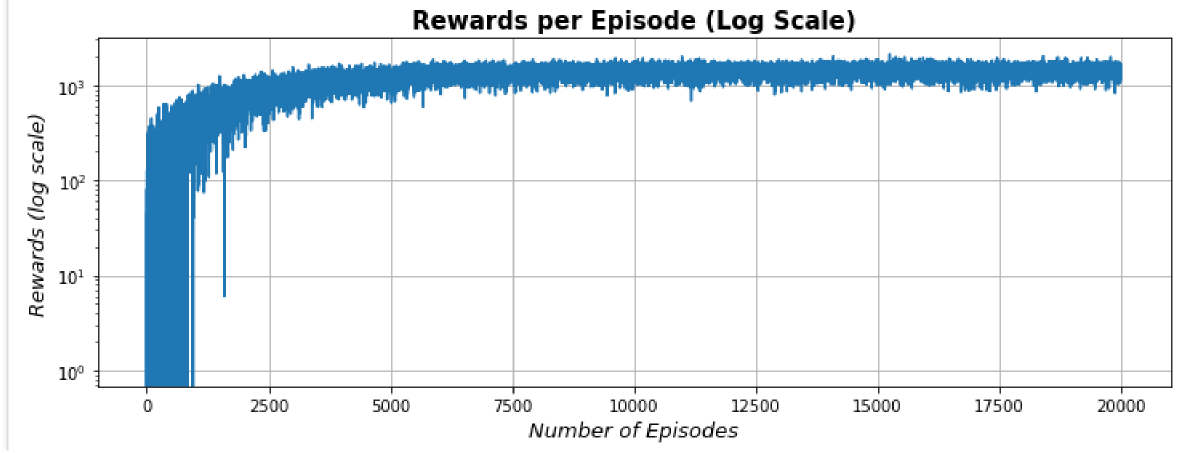
```
Tracking Convergence for state-action pair 2: State (1,2,3), Action (1,2)
```



While training our model we tracked the Q-values for state-action pair for (2,4,6) ;(2,3) and (1,2,3) ;(1,2) and plotted after completion of model training. The plots depict that our model has reached convergence by 4000 iterations since the change in Q-values almost became null.

Revenue Generation





Based on our assumptions that the expenditure of cab driver per hour of drive is 5 units and the earnings per hour of dropping the customer from pick up point to drop location is 9 units, our model is able to generate a monthly revenue of 1700-1800 units.

Conclusion and Future Works

This project successfully proposed a novel method falling within deep reinforcement learning to pursue the maximum profits of cab drivers. By formulating the driver's behavior to pick up the most rewarding action to the MDP process, we calculated state-action function value in the Q-table and used it as a reference for training our neural network model. The project uses the historic data to recommend the optimal choice of idling and serving to drivers at a particular location and time.

The future works can be working on the following parameters: -

- We assumed that the driver will not take any break for personal purposes or refueling and that he will work constantly 24*7. This is however not similar to real life scenario.
- Our project did not include the possibility of customer cancelling the request while the driver is reaching the pick-up point from his current location. Our model will fail to predict the best action the driver should take then.
- The locations can be mapped to real life locations using Google Maps API.

References

1. https://www.mdpi.com/2071-1050/12/21/8883/htm
2. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/
3. https://www.geeksforgeeks.org/deep-q-learning/
4. https://www.youtube.com/watch?v=OYhFoMySoVs
5. Reinforcement Learning: An Introduction - Richard S. Sutton and Andrew G. Barto