ARTIFICIAL INTELLIGENCE PROJECT

ON

RL BASED SYSTEM FOR ASSISTING CAB DRIVERS

SUBMITTED BY

LAKSHMI PRASANNA MANDADI(00748298)
ESWAR MUNDRU(00763874)
SURENDRA KAMMA(00693588)

SUBMITTED TO

Dr. SHIVANJALI KHARE

TABLE OF CONTENTS:

INTRODUCTION:

In this competitive world, Most drivers get healthy number of rides from customers throughout the day. But with the recent hikes in gas and electricity prices, many drivers complain that their profits are not increasing although their revenue is increasing. It is very important to choose the right rides for drivers which helps to maximize their profits throughout the day. In our project, we are using Reinforcement Learning for assisting cab drivers to maximize the profits for them.

PROJECT OBJECTIVE:

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

For example, say a driver gets three ride requests at 3 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.

APPROACH:

In this project, we need to create the environment and an RL agent that learns to choose the best request. We need to train our agent using Deep Q-learning (DQN).

Goals:

- **Create an environment:**

  The ‘Env.py’ file is the "environment class" - each method (function) of the class has a specific purpose.

- **Build an Agent:**

  Build an agent that learns to pick the best request using DQN. We can choose the hyperparameters (epsilon (decay rate), learning-rate, discount factor etc.) of our choice.

- Check for convergence of Q-Values by sampling few state-action pairs and plotting their Q-Values along episodes.

  Training depends purely on the epsilon-function we choose. If it decays fast, it won't let our model explore much and the Q-values will converge early but to suboptimal values. If it decays slowly, our model will converge slowly.

ASSUMPTIONS:

- The taxis are electric cars. It can run for 30 days non-stop, i.e., 24*30 hours.

  Then it needs to recharge itself. If the cab driver is completing his trip at that time, he will 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 cab driver crosses 24*30 hours.
- There are only 5 locations in the city where the cab can operate.
- All decisions are made at hourly intervals. We won't consider minutes and seconds for this project. So for example, the cab driver gets request at 3 PM then at 4 PM and so on. He can decide to pick among the requests only at these times. A request cannot come at 3.30 PM.
- 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.

Reinforcement Learning:

Reinforcement Learning is a subset of machine learning. It enables an agent to learn through the consequences of actions in a specific environment.

It differs from other forms of supervised learning because the sample data set does not train the machine. Instead, it learns by trial and error. Therefore, a series of right decisions would strengthen the method as it better solves the problem. Reinforced learning is like what we humans have when we are

children. We all went through the learning reinforcement — when you started crawling and tried to get up, you fell over and over, but your parents were there to lift you and teach you.

Challenges:

- Reinforcement learning's key challenge is to plan the simulation environment, which relies heavily on the task to be performed.
- Transferring the model from the training setting to the real world becomes problematic.
- Scaling and modifying the agent's neural network is another problem.
- Another difficulty is reaching a great location — that is, the agent executes the mission as it is, but not in the ideal or required manner.

Reinforcement is done with rewards according to the decisions made; it is possible to always learn continuously from interactions with the environment. With each correct action, we will have positive rewards and penalties for incorrect decisions. In the industry, this type of learning can help optimize processes, simulations, monitoring, maintenance, and the control of autonomous systems.

Markov Decision Process:

In mathematics, a Markov decision process (MDP) is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming.

A Markov decision process is a 4-tuple **(S, A, P_a, R_a**), where:

- S is a set of states called the state space.
- A is a set of actions called the action space (alternatively, A_s is the set of actions available from state s).

- P(s, s')= Pr(s_t+1=s'|s_t=s, a_t=a) is the probability that action a in state s at time t will lead to state s' at time t+1.
- R_a(s, s') is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a.

The state and action spaces may be finite or infinite, for example the set of real numbers. Some processes with countably infinite state and action spaces can be reduced to ones with finite state and action spaces. A policy function pi is a (potentially probabilistic) mapping from state space to action space.

Deep Q-Learning:

Q-learning is a simple yet quite powerful algorithm to create a cheat sheet for our agent. This helps the agent figure out exactly which action to perform. But what if this cheat sheet is too long? This presents two problems:

- First, the amount of memory required to save and update that table would increase as the number of states increases.
- Second, the amount of time required to explore each state to create the required Q-table would be unrealistic.

In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning & deep Q-learning is wonderfully illustrated as below.

Code Explanation:

Environment Class:

A reinforcement learning task is about training an agent which interacts with its environment. The agent arrives at different scenarios known as states by performing actions. Actions lead to rewards which could be positive and negative.

The agent has only one purpose here – to maximize its total reward across an episode. This episode is anything and everything that happens between the first state and the last or terminal state within the environment. We reinforce the agent to learn to perform the best actions by experience. This is the strategy or policy.

This is the "environment class" - each method (function) of the class has a specific purpose. Initialize the hyper parameters

- m =5, number of locations ranges from 0...m-1
- t =24, number of hours, ranges from 0 .... t-1
- d =30, number of days, ranges from 0 ... d-1
- C =5, Cost Per hour for fuel and other costs
- R =9, Reward per hour revenue from a passenger

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

We have 2 architectures of DQN,

- We pass only State as input
- We pass State and Action as input

The Architecture 1 (only State as input) performs better than Architecture 2 because we will get Q(s, a) for each action, so you have to run the NN just once for every state. Take the action for which Q(s, a) is the maximum.

Next State function:

Takes state and action as input and returns next state with considering below conditions.

- Driver refuse to request.
- Cab is already at pick up point.
- Cab is not at the pickup point.

Reward function:

Assessment requires to determine what action is to be taken to minimize loss and maximize benefits. The reward, r(s, a), in our system for taking an action $a \in A$ at a given state $s \in S$ is computed as follows.

Cab Driver DQN Agent:

In Agent class we need to work on below functions are

- Assigning hyperparameters
- Creating a neural-network model.
- Define epsilon-greedy strategy.
- Appends the recent experience state, action, reward, new state to the memory.
- Build the DQN model using the updated input and output batch.

Hyperparameters:

We can tweak these parameters for better performance.

```
self.discount_factor = 0.95
self.learning_rate = 0.01
self.epsilon = 1
self.epsilon_max = 1
self.epsilon_decay = -0.0001
self.epsilon_min = 0.00001
```

Neural Network Model:

Using keras we build a sequential model by adding dense layers.

We have provided state as input at the first layer with relu nonlinear activation function and then added hidden layers for better learning with relu activation function.

Here, we are using Adam optimizer which uses epsilon greedy policy and learning rate to improve the weights and bias to minimize mean square error.

Build the DQN model:

Appends the recent experience state, action, reward, new state to the memory with updated input and output batch.

EVALUATION:

Model continuously update strategies to learn a strategy that maximizes long-term cumulative rewards.

Below two are the performance metrices for our model.

- Q-Value convergence.
- Rewards per episode.

RESULTS:

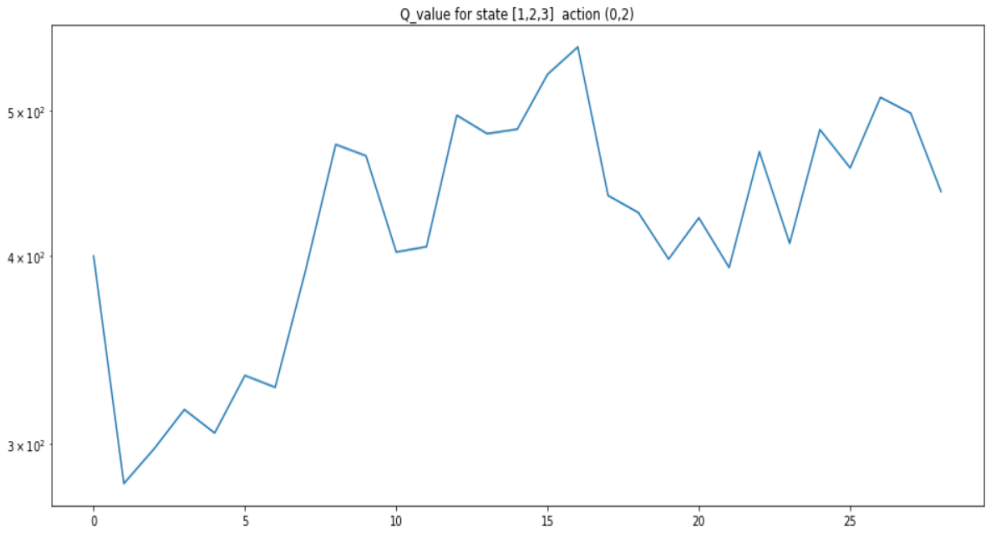
At initial stage, rewards are not better since our agent didn't learn since it's have less experience.

```
1/1 [==============================] - 0s 13ms/step
episode 0, reward -535.0, memory_length 144, epsilon 0.99999 total_time 734.0
Saving Model 0
```

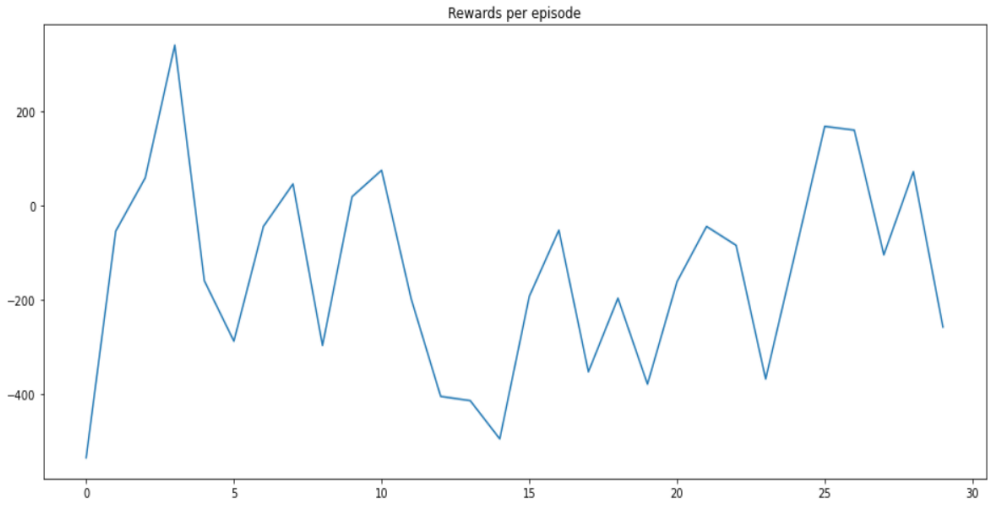
After some episodes, our agent got the positive rewards, as the episode increases the rewards for the agent also increases.

```
1/1 [==============================] - 0s 28ms/step
episode 10, reward 58.0, memory_length 1518, epsilon 0.9950025290678904 total_time 721.0
Saving Model 10
```

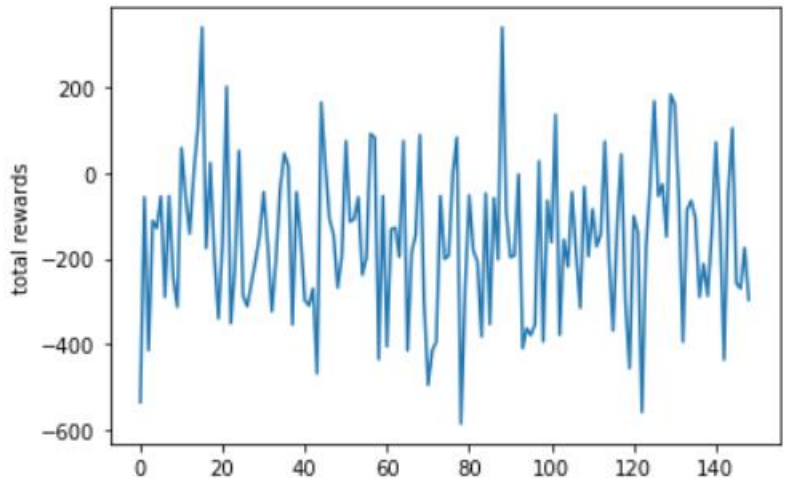
Q- Value Convergence:



Rewards per episode:



Total Rewards:



CONCLUSION:

We have modeled a RL-Based system agent using Deep Q-Learning Network, which can help cab drivers to maximize their profits by improving agent's decision-making process. We have plotted Q-value convergence and rewards per episode to understand the model performance. As we increase the number of rides in the system, the collected reward increases.