ARTIFICIAL INTELLIGENCE PROJECT ON

DECISION MAKING FOR CAB DRIVERS
Using Reinforcement Techniques

SUBMITTED BY

MINNU VARGHESE (00922453)

SRI LAKSHMI TANIKONDA (00873237)

VENKATA SAI LIKITH YEDIDA (00870615)

SUBMITTED TO

Dr. SHIVANJALI KHARE

TABLE OF CONTENTS:

INTRODUCTION:

In the highly competitive taxi industry, drivers often manage to secure a good number of rides throughout the day. However, many drivers find that despite increased revenue, their actual earnings aren't improving, especially due to the recent gas and energy costs spikes. Drivers need to make informed decisions to maximize their daily earnings by choosing the most profitable rides. This project leverages reinforcement learning to aid cab drivers in boosting their income. enhancing decision-making skills and optimizing routes in the fast-paced urban transport sector can greatly improve customer satisfaction and overall efficiency. The Cab Driver Assistance System uses reinforcement learning, a type of artificial intelligence, to offer smart decision support for cab drivers.

APPROACH:

This project aims to create an environment and a reinforcement learning (RL) agent that can choose the best requests. For training our agent, we'll be utilizing Deep Q-learning (DQN).

OBJECTIVES:

Our project aims to develop an RL-based algorithm designed to assist cab drivers in maximizing their profits by enhancing their on-field decision-making process.

- By using reinforcement learning methods, the system can learn and recommend optimal routes based on historical data, user preferences, and current traffic conditions. This will ensure ideal route planning.
- Dynamic Decision Support: Equip taxi drivers with a real-time decision-making tool that considers various factors such as fluctuating customer demand, road congestion, and weather conditions.
- Adaptive Learning: Develop a system that continuously learns from user feedback and the actions of taxi drivers, adapting its strategies to the dynamic urban environment.

For instance, imagine a driver receiving three ride requests at 3 PM. The first request offers a long-distance ride with a high fare, but it ends at a location where securing another ride for several hours is improbable. The second request finishes in a more favorable location but involves a minor detour for pickup, increasing fuel expenses. The third option, a medium-distance ride, seems to be the best choice since it not only likely secures another subsequent ride but also avoids most of the traffic.

GOALS:

- **Create an Environment:**
  The "environment class" is defined in the "Env.py" file. Each method (function) within this class is designed to perform a specific task.

- **Build an Agent:**
  Develop an agent that utilizes DQN (Deep Q-learning) to determine the best request. We have the flexibility to choose our hyperparameters, such as the learning rate, discount factor, and epsilon (decay rate). To assess if the Q-Values are converging, plot the Q-Values of a small sample of state-action pairs across episodes.

Training is heavily influenced by our choice of the epsilon function. If it decays too quickly, our model will have limited exploration and the Q-values will converge early, but not to optimal levels. Conversely, if it decays too slowly, the model will converge more gradually.

COMPONENTS:

- The environment model acts as an effective training ground for reinforcement learning algorithms by replicating city traffic patterns, roadways, and potential obstacles.
- Reinforcement learning methods, such as Q-learning and Deep Q Network (DQN), assist the system in determining the best rules through trial and error.
- User Interface (UI): Develop an intuitive UI that provides real-time recommendations to taxi drivers, helping them make informed decisions. The interface should also facilitate user feedback to support continuous learning.

ASSUMPTIONS:

- Electric taxis can operate continuously for 30 days, or 720 hours (24 hours * 30 days). After this period, they must recharge. The terminal condition occurs when the taxi completes 720 hours of operation, regardless of the number of rides taken within the month.
- The cab can only operate in five designated areas within the city.
- Decisions are made hourly. Minutes and seconds are not considered in this project. For example, a cab driver may receive requests at 3 PM, 4 PM, etc., and can only choose which request to accept during these specific hours. Requests cannot be made at 3:30 PM.

Travel time between locations is measured in whole hours and depends on traffic conditions, which are influenced by the time of day and the day of the week.

BENEFITS:

- **Efficiency**: Suggest optimal routes to minimize travel time and fuel usage.
- **Customer Satisfaction**: Reduce wait times and ensure reliable, timely service to enhance the overall customer experience.
- **Adaptability**: Continuously update the system to stay in sync with changing city dynamics and traffic conditions.

CODE EXPLANATION:

Environment Class:

The objective of a reinforcement learning task is to train an agent to interact effectively with its environment. By performing actions, the agent transitions through various states. These actions can result in either positive or negative rewards.

The agent's main aim is to maximize its cumulative reward throughout an episode. An episode encompasses everything that happens in the environment from the initial state to the terminal state. We encourage the agent to gain experience and learn to execute the optimal actions. This forms the basis of the approach or guideline.

This corresponds to the "environment class," where each function or method within the class serves a specific purpose. Begin by setting the hyperparameters to their initial values.

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

To input data into the neural network (NN), convert the state into a vector format, which results in a vector of size m + t + d.

We have two architectures of Deep Q-learning (DQN):

1. **Architecture 1**: Uses only the state as input.
2. **Architecture 2**: Uses both state and action as input.

Architecture 1 outperforms **Architecture 2** because it requires executing the NN only once for each state. This is efficient because it produces the Q-values Q (s, a) for all possible actions. You then choose the action with the highest Q (s, a).

Next State function:

Takes state and action as input and returns the next state considering the following conditions:

- **Driver refuses to request**: If the driver rejects the ride request.
- **Cab is already at the pickup point**: If the cab is already at the location of the passenger.
- **Cab is not at the pickup point**: If the cab needs to travel to the passenger's location.

Reward function:

Evaluation involves identifying the optimal action to minimize losses and maximize benefits. In our system, the reward, r (s, a), for taking an action $a \in A$ in a given state $s \in S$ is calculated as follows:

Cab Driver DQN Agent:

In the Agent class, we need to focus on the following functions:

- **Assigning Hyperparameters**: Set initial values for learning rate, discount factor, epsilon (exploration rate), and other necessary parameters.
- **Creating a Neural-Network Model**: Design and initialize the neural network architecture for the agent.
- **Defining Epsilon-Greedy Strategy**: Implement the epsilon-greedy strategy to balance exploration and exploitation during training.
- **Appending Recent Experience**: Store the latest state, action, reward, and new state in memory for experience replay.
- **Building the DQN Model**: Train the DQN model using the updated input and output batches based on the experiences stored in memory.

Hyperparameters:

We can adjust these parameters to enhance 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 construct a sequential model by incorporating dense layers.

To enhance learning, we add hidden layers with a ReLU activation function after providing the state as input to the initial layer, which also uses a ReLU non-linear activation function.

We employ the Adam optimizer, which utilizes an epsilon-greedy policy and adjusts the learning rate to optimize the weights and biases, thereby minimizing the mean square error.

Build the DQN model:

Stores the recent experience, including state, action, reward, and new state, in memory, and updates the input and output batch accordingly.

EVALUATION:

To discover a strategy that maximizes long-term cumulative benefits, the model continually refines its tactics.

Here are two performance metrics for our model:

- **Q-Value Convergence**: Evaluate if the estimated Q-values are stabilizing over time.
- **Rewards per Episode**: Measures the total rewards accumulated in each episode, indicating the model's effectiveness in achieving its objectives.

RESULTS:

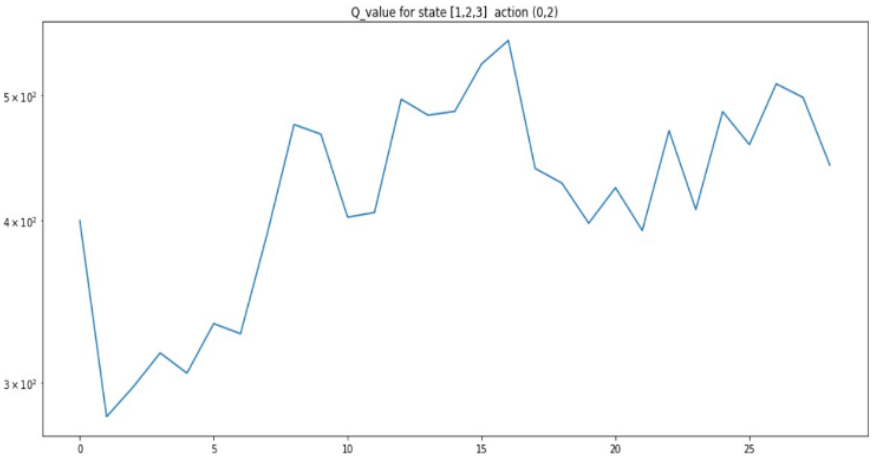
Initially, due to the agent's limited experience, its learning was insufficient, resulting in less effective incentives.

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

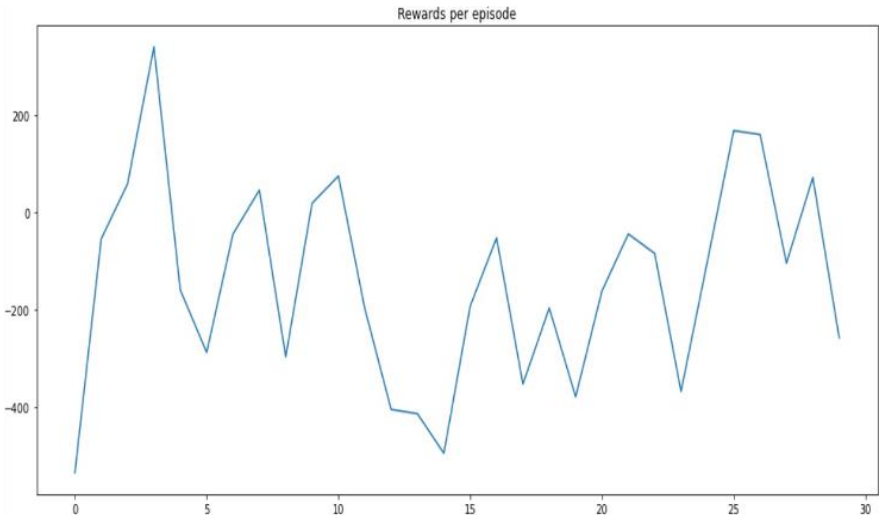
As the episodes progressed, our agent started receiving positive rewards. With each subsequent episode, the rewards for the agent continued to increase.

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

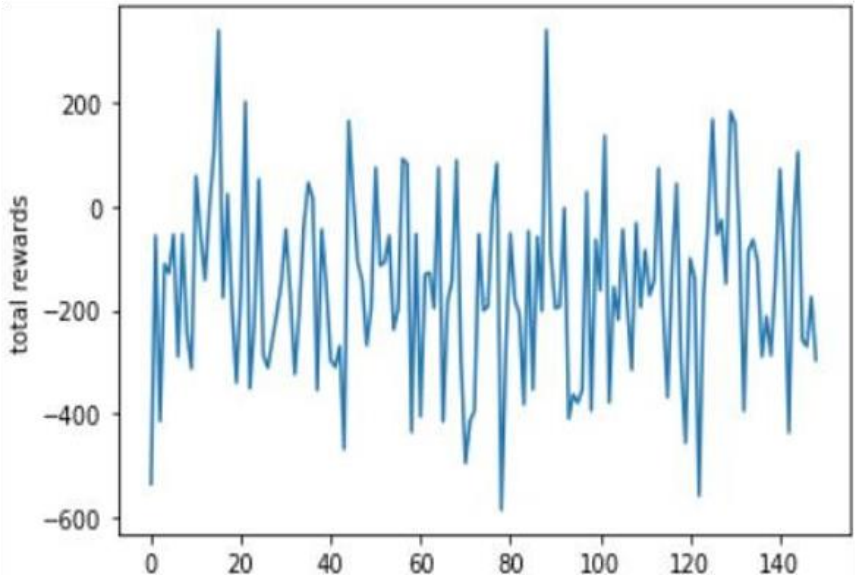
Q-VALUE CONVERGENCE:



REWARDS PER EPISODE:



TOTAL REWARDS:



CONCLUSION:

We've developed an RL-based system agent using the Deep Q-Learning Network to enhance taxi drivers' decision-making and optimize their profits. To illustrate the model's success, we have demonstrated Q-value convergence and rewards per episode. As more rides are incorporated into the system, the cumulative reward increases. The Cab Driver Assistance System, powered by reinforcement learning, equips taxi drivers with smart tools to navigate cities more efficiently and provide better services to customers.