CSCE 642 Deep Reinforcement Learning
RL for Driver Revenue Optimization

Anirith Pampati
135001434

Priyanka Askani
234009293

1. Motivation

In the dynamic ride-sourcing market, cab drivers are usually faced with fluctuating demand, variability in traffic, and competition that forces them to make suboptimal decisions, compromising profitability. Most of the current systems lack personalized, real-time decision support, hence leading to inefficiencies. This research will seek to develop an RL-based system that empowers drivers with intelligent, data-driven strategies, enhancing their ability to make informed decisions, optimize routes, and ultimately increase their revenue while improving overall service efficiency.

2. Introduction

Recent breakthroughs in reinforcement learning have shown the capability to solve complex, real-world problems that inherently require sequential decision-making. Cab drivers in ride-hailing must make constant decisions in a continually changing environment regarding ride acceptance and routing that greatly impact profitability. To address this challenge, we proposed creating an RL-based system that can assist cab drivers in optimizing their revenue using intelligent, data-driven decisions.

Our project revolves around designing a decision-making model for drivers using parameters such as pickup and drop-off locations, time of day, and future opportunities to make rides. This method aims at optimizing earnings for drivers, system-wide efficiency in ride-hailing, and, to an extent, driver retention through usable insights. We did this by applying deep reinforcement learning methods in an MDP setting to model the sequential decisions in this problem domain.

We built upon this RL-based implementation and re-engineered the model to incorporate realistic, dynamic traffic data and probabilistic ride requests. We faced challenges in state transitions and time-varying reward functions. The baseline approach involves the utilization of a DQN agent for driver decision optimization, and enhancements are further made with Double DQN architectures.

Our project utilizes some core concepts of reinforcement learning, including MDPs, Q-learning, and policy gradient methods that are central to themes discussed in this course. We extend the theory by implementing advanced techniques like Deep Q-Networks (DQN), based on the foundational concepts covered in class, toward developing solutions to optimize cab driver revenues within dynamic environments.
Our implementation and video are available at https://github.com/Askani-Priyanka/DRL and https://www.youtube.com/watch?v=-1YU0OJqRpU.

3. Related Work

In this section, we compare our work with several state-of-the-art works in the realm of machine learning, reinforcement learning, and revenue optimization for transportation systems. These references provide an idea of architectural frameworks, model-based RL methodologies, and the application of sequential decision-making in dynamic environments. We analyze the core concepts and innovations presented in these studies and highlight how our project aligns with or deviates from their approaches to address unique challenges in driver revenue optimization.

3.1 A Better Match for Drivers and Riders [1]: Reinforcement Learning at Lyft

This paper presents similar work in optimizing ridesharing services through reinforcement learning in real-time decision-making for profit maximization, matching drivers to riders. Yet, there are differences in this work regarding algorithm, objective, environment, state space, and action space. Lyft's algorithm generates regular matches efficiently by predicting driver earnings in the future, while our work leverages a Deep Q-Network for profit maximization. Finally, Lyft's algorithm runs in real time, taking into account the actual-time availability of drivers and riders' demand, whereas our work considers a simulated period of 30 days with location and time-of-day information, including day of week. Key takeaways from the Lyft paper: real-time optimization works effectively, scalable, significant revenue increase ($30M/year).

https://pubsonline.informs.org/doi/10.1287/inte.2023.0083

3.2. Taxi Revenue Optimization with Deep Q-Learning and Enhanced Data Visualization [2].

This paper discusses how to maximize the profit of taxi drivers using DRL, which aligns with the goal of our project. Both utilize DQN for making decisions that aim at maximizing profits. There are differences in environment, state space, action space, and reward functions. The default TaxiTripEnv in the paper contrasts with our 5-location, 24-hour, 7-day setting. The methodological differences are epsilon-greedy exploration - exploitation and Google Colab training. Key findings: 133.33% increase in reward, effective selection of a trip, and growth in profit up to 2.8 times. It could also be directed to transfer learning, cooperative optimization, and real-world deployment, showing that DRL has great potential in taxi service and beyond.

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10574699

3.3. Augmenting Decisions of Taxi Drivers through Reinforcement Learning for Improving Revenues [3].

This paper is a similar study to our project in the objective of its maximization of revenue for a taxi driver by reinforcement learning, where real-world data is used to estimate the best source locations for customer pickups. Both projects take a driver's perspective and thus aim at maximizing earnings by directing drivers to the places of high demand rather than merely trying to improve customer service. However, this paper relies on supervised feature learning from historical trajectory logs by annotating activities such as roaming or waiting at taxi stands. This is intrinsically a more complicated action space and requires accurate feature engineering. Our approach simplifies the environment with a pre-computed time matrix and probabilistic ride requests, providing structured states and actions based on pickup-drop-off pairs rather than free-roaming activities. Moreover, our work introduces a reward model factoring in operational costs such as battery consumption, unlike this study that focuses primarily on location-based earnings. This dynamic contrast is what our project has focused on.

https://ojs.aaai.org/index.php/ICAPS/article/view/13846/13695

3.4. Short‑term prediction of demand for ride‑hailing services: A deep learning approach. Transportation [4].

This paper is related to our project because the aim is to investigate optimization of ride-hailing services; this work falls precisely on driver allocation through demand estimation. However, UberNet uses CNN for spatial and temporal complexities in demand estimation, focusing its attention on short-term demand forecasting, whereas our project adopts an RL approach: guide the drivers to move them toward maximizing their long-term revenue based on the probabilistic ride requests. Indeed, UberNet's multivariate framework embeds a wide variety of features including weather, demographics, and built environment factors, which allows them to capture the complex motivators of travel behavior and enhance the accuracy of prediction. By contrast, our model adopts DQN within a structured MDP framework that can simulate decisions over a 30-day period with a reward structure based on revenue and operational costs such as battery usage. In contrast, this work focuses on sequential decision-making with more emphasis on the RL model for improving profitability over an elongated period, while demand forecasting with a CNN in UberNet provides real-time, accurate insights about demand.

https://link.springer.com/article/10.1007/s42421-021-00041-4

3.5. An intelligent framework to maximize individual taxi driver income [6].

This paper is also closely related to our project. It uses the Deep Q-Network framework to help individual drivers make appropriate decisions in terms of order selection and repositioning to maximize income. In this study, taxi operations were formulated as a Markov Decision Process, just like in our project, where it did a simulation of drivers in a multi-agent environment; in other words, each agent, or driver, learns strategies themselves to maximize daily income. While this also involves a DQN approach in our project, we restrict ourselves to decision modeling over a simulated 30-day period, adding an additional cost-based reward structure considering even factors like battery consumption. The approach of this paper has been to focus on real-time, short-term gains by recommending orders within specific fare ranges that optimize the immediate income, while our project explores the possibility for drivers making sequential, long-term, profit-oriented decisions. This has also let the application of multi-agent simulations in this scenario, allow for a strategic layer based on the interaction with other drivers, something different from the simulated environment of our project based on the independent decision of each driver.

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10407856

3.6. Dynamic pricing in Ride-Hailing intelligent transportation systems by using Deep Reinforcement Learning [7].

This study also uses reinforcement learning for optimizing driver revenue in ride-hailing systems, mainly through dynamic pricing for the balance between supply and demand. Both aim at an increase in the earnings and a decrease in passenger waiting time by taking real-time, data-driven decisions. However, this paper adopted the DSARSA algorithm, which copes with the complicated state-action situation, while using the MARCOS methodology for ranking models with multi-attributes. On the other side, the model we used in our project is a simpler Deep Q-Network model that considers a time matrix and a reward function including battery costs, thus focusing more on route optimization and efficiency. This also involves external conditions, such as weather which though adding flexibility, is different from our approach of using fixed sets of conditions. Cumulatively, these differences underline our focus on optimization of specific actions within structured scenarios and emphasize dynamic pricing for variable conditions in the present study.

https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4746254

3.7. Good or Mediocre? A Deep Reinforcement Learning Approach for Taxi Revenue Efficiency Optimization [8].

"Intelligent Framework to Maximize Individual Taxi Driver Income" shares a great similarity with our work: Deep Reinforcement Learning used to optimize taxi driver income, keeping in mind spatio-temporal facts which concentrate on maximum profit. It differs in approach, environment, features, and objectives, hence the paper applies Markov Decision Process formulation with Edge-DQN and has historical and real data from Changsha City while putting an emphasis on individual driver benefits. Both papers demonstrate Deep Reinforcement Learning's capability in maximizing drivers' income, and highlight the importance of spatial-temporal features. Future work may include combining MDP with DQN, transfer learning, and even real-world deployment. Strong points of our current work include reinforcement learning, profit maximization objective, and flexibility of simulations, while possible improvements might involve adding demand and average fare features, the study of MDP formulation and Edge-DQN, and incorporating more optimizations and edge data.

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9153065

3.8. Improving Taxi Revenue using Reinforcement Learning [9].

The work presented in this paper and our work are similar in that both apply reinforcement learning to the optimization of taxi driver income, considering profit maximization and spatio-temporal factors. The differences between these two are reflected in the approach used, environment, and features. Q-learning was applied in the paper, with simulated data and features related to cruising trajectory. Meanwhile, in our work, Deep Q-Networks are applied with historical data. Both studies illustrate the efficiency of reinforcement learning in revenue maximization and give importance to spatial temporality. The key takeaways will be the potential of reinforcement learning, the effectiveness of Q-learning, and the importance of space-temporal considerations. Future works involve incorporating Q-learning with DQN, transfer learning, and real-world deployment. Our contributions include the application of deep reinforcement learning, profit maximization objective, and flexibility in simulated environments, while potential improvements include incorporating the features of the cruising trajectory, studying Q-learning, and integrating several techniques of optimization.

3.9. An application of a deep Q-network-based dynamic fare bidding system to improve the use of taxi services during off-peak hours in Seoul. Sustainability [10].

This paper and our work have much in common in using Deep Reinforcement Learning to optimize the profits of taxis based on the implementation of dynamic pricing. However, the approach, environment, and objective are different: multi-agent Deep Q-Network, Seoul taxi data, and real-time fare bidding for the paper, while Deep Q-Networks and simulated environments that target individual driver income maximization are used in our work. Both works highlight how effective Deep Reinforcement Learning is to optimize taxi operations, emphasizing spatial-temporal importance. Key takeaways include the dynamic fare bidding potentials, consideration of supply-demand imbalance, and value of waiting time. These include multi-agent DQN combined with DQN, transfer learning, and real-world deployment.

https://www.mdpi.com/2071-1050/13/16/9351

3.10. A Deep Q-Learning Network Based Reinforcement Strategy for Smart City Taxi Cruising [11].

This work is closely related to our project because both of them want to improve the income of taxi drivers using an RL-based system that can optimize the process of finding passengers. Similar to our work, this paper employed DQN to develop a data-driven strategy guiding the drivers to areas with high passenger demand by using real taxi order data and minimizing their idle time. The current study mainly investigates the cost reduction of passenger search by comparing the RL approach with a random-walk strategy; our project extends the reward structure toward inclusion of revenue and operational cost, such as battery consumption, for a more holistic view of profitability. We also include temporal features such as time of day and day of week to further fine-tune the decisions made by the RL model. This paper applies its strategy over general time segments. The differences in our project lie in the broader goal of balancing various cost factors toward sustaining drivers' earnings over the long haul.

https://link.springer.com/chapter/10.1007/978-981-16-5188-5_5

4. Implementation Strategies

In this section, we built three different architectures to address the challenge at hand, each with the incorporation of neural networks to optimize decision-making in a framework of reinforcement learning. These represent the latest developments in policy optimization methods and value-based learning methods. Below is a detailed explanation of the implementation methodologies of each of these architectures:

4.1 Policy Gradient Architecture (PGA)

The Policy Gradient Architecture directly optimizes the policy, and this is performed by maximizing the expected cumulative reward. A neural network model will approximate the probability distribution of taking an action given a state. The methodology includes the following items:

4.1.1 Model Structure

1. **Input Layer**: Takes in the representation of the state as a vector of size state_size.
2. **Two Hidden Layers**: Dense layers with 200 and 150 units, respectively, with the ReLU activation function.
3. **Output Layer**: A softmax layer that outputs a probability distribution over all possible actions, with size action_size.

4.1.2 Training Process

1. **Storing Transitions**: The agent stores transitions as tuples of (state, action, reward).
2. **Discounted Rewards**: Rewards are first transformed using the discount factor "γ" to make the algorithm focus on immediate rewards rather than rewards far in the future.
3. **Policy Update**: The discounted rewards are normalized to help stabilize training.
4. The policy network is trained with a categorical cross-entropy loss, weighted by the computed rewards.

4.1.3 Objective

The goal of policy gradient methods for reinforcement learning is to directly optimize the policy by adjusting its parameters to maximize expected cumulative rewards, enabling agents to learn optimal behaviors in complex environments.

4.2. Deep Q-Network Architecture 1 (DQN1)

The DQN1 follows a value-based learning approach, where the agent learns to predict the Q-value corresponding to a particular state-action pair. In this architecture, every possible action requires a forward pass, so the model can be very computationally expensive.

4.2.1 Model Structure

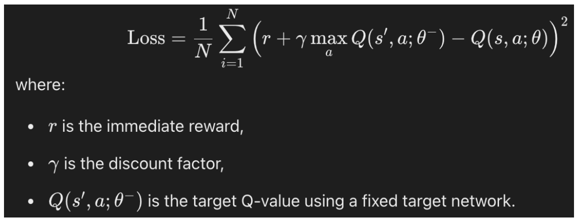
1. **Input Layer**: This is a layer that merges the state and action vectors into one input.
2. **Hidden Layers**: These process the combined input to obtain features expressing Q-value dynamics.
3. **Output Layer**: Produces a single scalar value representing the Q-value for the input (state, action) pair.

4.2.2 Training Process

1. **Experience Replay**: The agent stores every transition, which consists of the current state, action taken, reward received, and the next state, in a replay memory buffer.
2. **Batch Sampling**: The agent samples, in regular intervals, random batches of transitions from the replay memory to decorrelate data and enhance learning efficiency.
3. **Q-Value Update**: The agent calculates, for every transition sampled, the target Q-value by adding up the immediate reward with the maximum Q-value of the next state, discounted by the target network.
4. **Loss Calculation**: The agent calculates the difference between the predicted Q-value and the target Q-value, which measures the loss and quantifies the accuracy in the network's predictions.
5. **Optimize Network:** The agent is doing the backpropagation with the help of gradient descent in minimizing the loss by adjusting weights of the neural network so that it can make increasingly better predictions.
6. **Target Network Update**: Periodically, the main network weights are copied to the target network, which provides consistent target values and stabilizes the training process.

4.2.3 Q-Learning Objective

The network minimizes the Temporal Difference TD error:



$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(r + \gamma \max_{a} Q(s', a; \theta^{-}) - Q(s, a; \theta)\right)^2$$

where:

- r is the immediate reward,
- γ is the discount factor,
- $Q(s', a; \theta^{-})$ is the target Q-value using a fixed target network.

4.3 Deep Q-Network Architecture 2 (DQN2)

The DQN2 provides an optimization over DQN1 by predicting the Q-values for all possible actions in one forward pass, significantly improving computational efficiency.

4.3.1 Model Structure

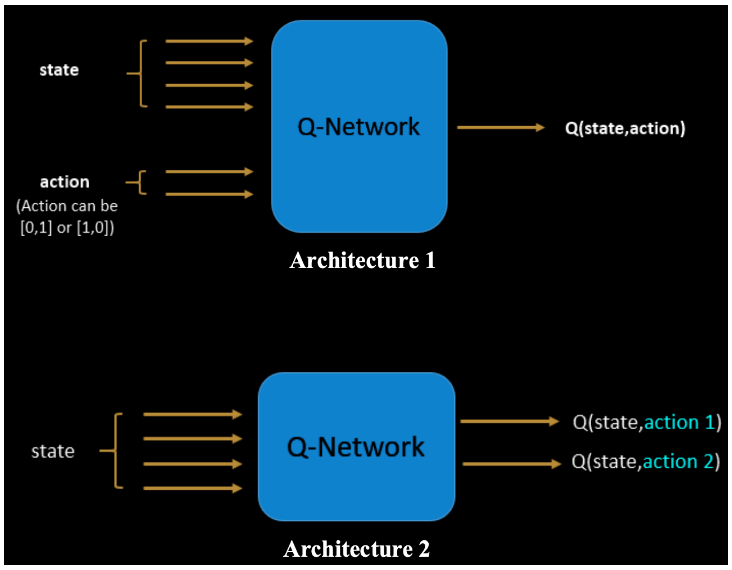
1. **Input Layer:** Takes as input the state vector.
2. **Hidden Layers**: Feature extraction of state-based features using dense layers.
3. **Output Layer**: Outputs a vector of Q-values over all possible actions of size action_size.
4. **Action Selection**: The action corresponding to the maximum Q-value is chosen.

4.3.2 Training Process

1. **Experience Replay**: The transitions are stored in a replay buffer to break the correlation between consecutive experiences.
2. **Batch Sampling**: Random sampling from the replay buffer will result in very diverse training data that promotes robust learning.
3. **Double Q-Learning for Q-Value Update**: The DQN2 applies Double Q-Learning to reduce the overestimation using the main network to select the best action and the target network to evaluate its value.
4. **Loss Calculation:** The loss is computed between the predicted Q-values and the more accurate target Q-values derived from Double Q-Learning.
5. **Optimize Network**: Using gradient descent, it updates the weights of the main network by minimizing the calculated loss.
6. **Target Network Update**: Target network's weights are periodically updated to match that of the main network, so the targets can be evaluated consistently.

4.3.3 Q-Learning Objective

The network minimizes the Temporal Difference TD error with a shared architecture for all actions.



4.4. Comparison and Selection

Each of the architectures offers unique advantages and trade-offs:

1. PGA excels in stochastic environments requiring direct policy optimization.
2. DQN1 offers a more focused view on particular state-action Q-values, but doesn't scale well computationally.
3. DQN2 balances this out by offering efficient action evaluations and retaining the benefits of value-based methods.

These architectures were implemented and tested in our implementation based on the performances of cumulative rewards, computational efficiency, and stability during training.

5. Code Structure and Environment Specifications

In this section, we will discuss our code structure and the environment we designed.

5.1 Code Structure

1. We maintained an .ipynb file, which contains all the three architectures we implemented. The evaluation part of the three architectures are also in the same file.
2. Time Matrix: TM.npy Time Matrix representing traffic.
3. H5 and Pickle Files: Saved Model Weights and Rewards and Tracked State Saved objects.
4. env.py: Environment containing the rules of the MDP.

5.2 State Space

Here, the state space is defined by the driver's current location along with the time components, an hour of the day, and a day of the week.
A state is defined by three variables:

S = X, T, D, where,
X represents a driver's current location.
T represents time component more specifically hour of the day.
D represents the day of the week.

Example: Number of locations: m = 6, Number of hours: t = 24, Days: d = 7.
Terminal state: The cab reaches his 30 days, hence the length of one episode is 30 days.

5.3 Action Space

Every hour, ride requests arrive from the customers in the form of the (pick-up, drop) location. Based on the current 'state', he needs to take an action which could maximize 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. General case: States do not necessarily result in an equal number of requests that the cab driver gets.

5.4 Transition Function

The transition function in this problem describes how the system moves from one state s=XiTjDk, representing the current location, time, and day, to a new state s′=XqTt′Dd′ after taking an action, a such as traveling from a pickup location to a drop-off location. The travel time between locations is looked up in a pre-calculated 4-dimensional time matrix, which factors in the start and end locations, the hour of the day, and the day of the week. This matrix reflects the variability in travel times based on traffic conditions and other factors. Once the travel time is determined, the next state is computed by updating the current time and day according to the travel duration. Thus, the transition function maps the current state and action to a future state by considering spatial and temporal changes, driven by the time matrix.

5.5 Reward Function

$$\mathbf{R(s) = R_k \times Time(p,q) - C_f \times (Time(p,q) + Time(i,p))}$$

R_k is the revenue earned per hour of driving. C_f is the cost of battery consumption per hour.
Time(p,q) is the travel time from the pickup location p to the drop-off location q, looked up from the time matrix. Time(i,p) is the travel time from the current location i to the pickup location p, also from the time matrix.

6. Results

In this section, we will discuss our final findings:

6.1 Cumulative Average Rewards Over 2,500 Episodes Arch-1 vs Arch-2:

Architecture	200	400	600	800	1000	1200	1400	1600	1800	2000	2500
Arch - 1	**<u>-0.08</u>**	-1.78	**<u>1.16</u>**	-0.43	1.61	**<u>2.24</u>**	-0.94	-0.11	-2.00	2.53	2.52
Arch - 2	-1.5	**<u>-1.06</u>**	0.86	**<u>0.6</u>**	**<u>2.25</u>**	1.76	**<u>2.8</u>**	**<u>1.71</u>**	**<u>3.4</u>**	**<u>4.07</u>**	**<u>6.008</u>**

The bold and underlined rewards represent the better model until that point of episode number.
****We can see clear domination of Arch - 2 over Arch - 2***

6.2 Success Rates(SR) Variation Over 2,500 Episodes Arch-1 vs Arch-2 in %:

Architecture	200	400	600	800	1000	1200	1400	1600	1800	2000	2500
Arch - 1	08.45	12.21	18.63	25.21	28.27	29.55	33.33	35.97	38.31	40.37	46.00
Arch - 2	20.39	25.68	34.44	42.69	48.85	54.45	58.52	63.14	66.66	69.61	62.49

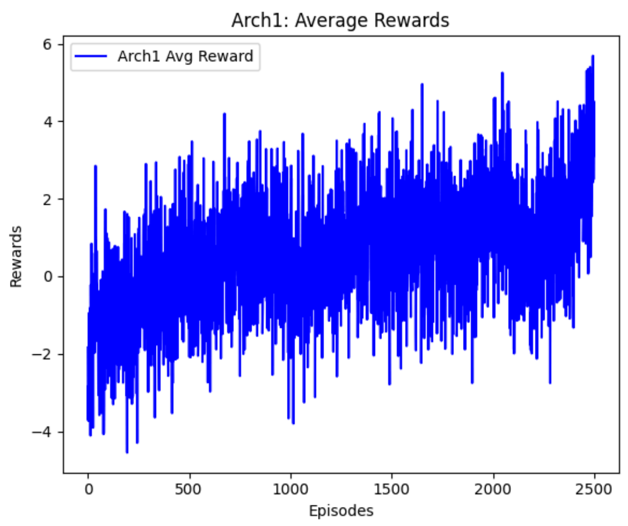
Success rate measures the proportion of successful episodes relative to the total episodes completed.
****SR of Arch-2 is always greater than Arch-1 from the start***

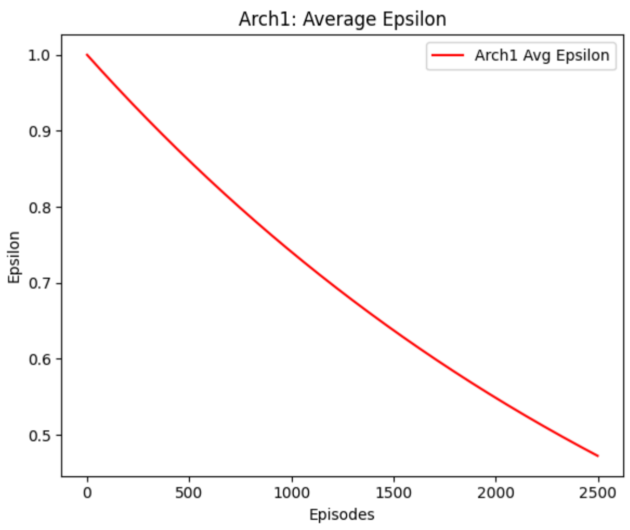
6.3 Final Success Rates of the DQN models:

Architecture	Success Rate
DQN Arch - 1 (2,500 episodes)	**46%**
DQN Arch - 2 (10,000 episodes)	**90.38%**

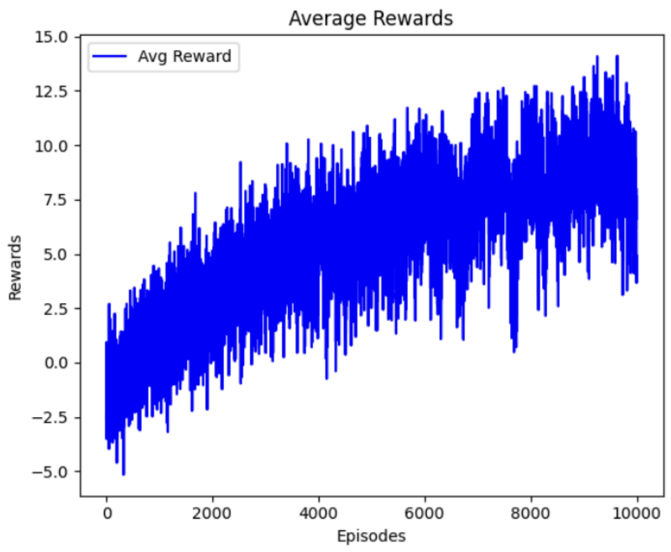
As the results of Arch-2 were promising we trained Arch-2 over 10,000 episodes.

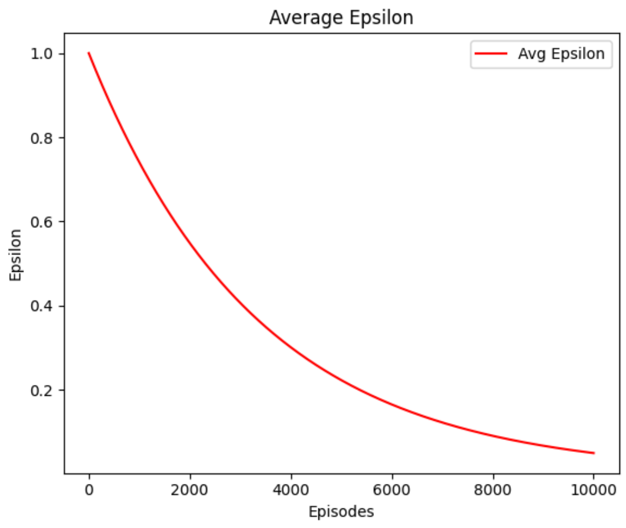
6.4 Convergence Plots of DQN Arch-1:





6.5 Convergence Plots of DQN Arch-2:





6.6 Episodic Rewards of the Policy Gradient:

Architecture	200	400	600	800	1000	1200	1400	1600	1800	2000	2500
PGA	6.00	4.93	4.26	2.70	5.48	5.95	3.80	4.77	4.75	6.42	5.35

Observations: We present the results for Policy Gradient separately from the DQN because DQN used average reward per episode as a metric, calculated using the trip count for each episode. In contrast, Policy Gradient updates are at an episodic level, and there is no concept of trip count, so we use episodic rewards as a metric, making direct comparison between the two metrics impossible. The reward per episode is quite low and differentiated, some outputs are as low as 1.95 for episode 2200, while others are slightly high, at 6.42 for episode 2000, therefore, the policy is still changing and has not stabilized yet. Policy Gradient methods are known to be of high variance, which may explain such inconsistent rewards.

7. Future Work and Limitations

1. **Integrating Real-World Data**: Use real-world data instead of simulated environments to improve practical applicability for solutions. Collaborate with domain experts to improve reward functions and state-transition functions using domain-specific metrics.

2. **Scaling to Larger State and Action Spaces**: In larger state and action spaces, the use of architectures such as Transformers for state encoding or multi-agent reinforcement learning systems can be explored for complex decision-making. Implement curriculum learning strategies that will progressively train the model in more challenging environments.

3. **Improving Generalization**: The current architectures focus on specific state-action spaces. The future might involve training these models in more diverse datasets and environments to improve generalization on unseen scenarios. Introduce meta-reinforcement learning that allows the model to adapt to new environments with minimal retraining.

8. Conclusion

The designs and evaluations were carried out in this project for three different architectures of Q-network to enhance decision-making in reinforcement learning tasks. From a comparative standpoint, we unveiled the trade-offs inherent to the three studied architectures among their computational efficiency, scalability, and learning performance. Results showed that DQN2 is superior in terms of rewards/success rate with the inclusion of computational advantages by predicting all Q-values simultaneously, while the DQN-1 doesn't have great success rates it has its own advantages such as flexibility and the policy gradient method provides a direct approach to selecting actions in continuous spaces. On the other hand, this study also showed the limitations regarding scalability, efficiency in exploration, and sensitivity to hyperparameters. In summary, the knowledge/algorithms acquired from the course work constitutes the basis for foundation of this project, which also helps in enabling applications that are more robust, scalable, and practical in real-world scenarios.

9. References

[1] Loncaric, S. Martin, V. Narasiman, Z. Qin, B. Richard, S. Smoot, S. Taylor, G. van Ryzin, D. Wu, F. Yu, and A. Zamoshchin, "A Better Match for Drivers and Riders: Reinforcement Learning at Lyft," Lyft, San Francisco, California, 94107.

[2] B. Balaji, A. T. Mithul Raaj, V. Harsath, R. R. Sai Arun Pravin, C. Rani, G. Aarthi, G. Aggarwal, and M. Rajesh Kumar, "Taxi Revenue Optimization with Deep Q-Learning and Enhanced Data Visualization," in Proc. IEEE.

[3] Tanvi Verma, Pradeep Varakantham, Sarit Kraus, Hoong Chuin Lau, "Augmenting Decisions of Taxi Drivers through Reinforcement Learning for Improving Revenues"

[4] Chen, L., Thakuriah, P. (Vonu), & Ampountolas, K. (2021). Short‑term prediction of demand for ride‑hailing services: A deep learning approach. Transportation, 48(6), 3233-3255. https://doi.org/10.1007/s11116-021-10175-y

[5] Dhamodar-DDR, "Optimal cab route prediction," GitHub. [Online]. Available: https://github.com/Dhamodhar-DDR/Course-Projects/tree/master/Optimal%20Cab%20route%20prediction

[6] Chen, F., Cai, H., & Wan, H. An intelligent framework to maximize individual taxi driver income. School of Industrial Engineering, Purdue University; Department of Industrial and Systems Engineering, North Carolina State University.

[7] Sajad Heydari, Elham Akhondzadeh Noughabi, "Dynamic pricing in Ride-Hailing intelligent transportation systems by using Deep Reinforcement Learning"

[8] H. Wang, H. Rong, Q. Zhang, D. Liu, C. Hu, and Y. Hu, "Good or Mediocre? A Deep Reinforcement Learning Approach for Taxi Revenue Efficiency Optimization," IEEE Transactions on Network Science and Engineering, vol. 7, no. 4, pp. 2819-2831, Oct.-Dec. 2020

[9] S. Subham, S. Singh, A. Sunil Kumar, F. Fatima, and G. Geetha, "Improving Taxi Revenue using Reinforcement Learning," in IETE – 2020 Conference Proceedings, International Journal of Engineering Research & Technology (IJERT), Bangalore, India, 2020.

[10] Cho, Y., Song, J., Kang, M., & Hwang, K. An application of a deep Q-network-based dynamic fare bidding system to improve the use of taxi services during off-peak hours in Seoul. Sustainability.

[11] Hua, Z., Li, D., Guo, W. (2021). A Deep Q-Learning Network Based Reinforcement Strategy for Smart City Taxi Cruising. In: Zhang, H., Yang, Z., Zhang, Z., Wu, Z., Hao, T. (eds) Neural Computing for Advanced Applications. NCAA 2021. Communications in Computer and Information Science, vol 1449. Springer, Singapore.