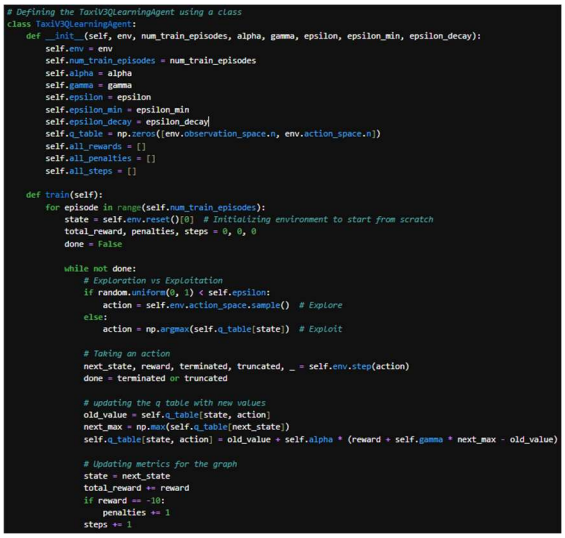
MODULE : 6ELEN018W - Applied Robotics
NAME : ABHINAVA SAI BUGUDI
ID : W1947458
LECTURER NAME : Dr Dimitris C. Dracopoulos

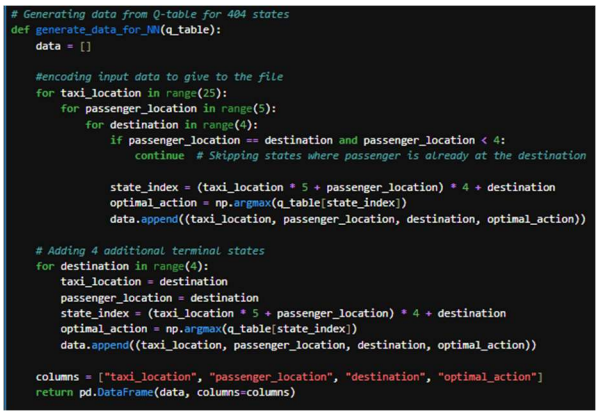
IMPLEMENTATION OF Q LEARNING ALGORITHM:



```python
# Defining the TaxiV3QLearningAgent using a class
class TaxiV3QLearningAgent:
    def __init__(self, env, num_train_episodes, alpha, gamma, epsilon, epsilon_min, epsilon_decay):
        self.env = env
        self.num_train_episodes = num_train_episodes
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_min = epsilon_min
        self.epsilon_decay = epsilon_decay
        self.q_table = np.zeros([env.observation_space.n, env.action_space.n])
        self.all_rewards = []
        self.all_penalties = []
        self.all_steps = []

    def train(self):
        for episode in range(self.num_train_episodes):
            state = self.env.reset()[0]  # Initializing environment to start from scratch
            total_reward, penalties, steps = 0, 0, 0
            done = False

            while not done:
                # Exploration vs Exploitation
                if random.uniform(0, 1) < self.epsilon:
                    action = self.env.action_space.sample()  # Explore
                else:
                    action = np.argmax(self.q_table[state])  # Exploit

                # Taking an action
                next_state, reward, terminated, truncated, _ = self.env.step(action)
                done = terminated or truncated

                # updating the q table with new values
                old_value = self.q_table[state, action]
                next_max = np.max(self.q_table[next_state])
                self.q_table[state, action] = old_value + self.alpha * (reward + self.gamma * next_max - old_value)

                # Updating metrics for the graph
                state = next_state
                total_reward += reward
                if reward == -10:
                    penalties += 1
                steps += 1
```

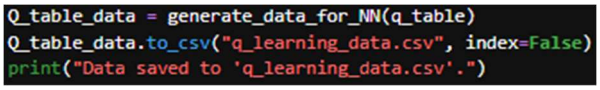
The Bellman equation was utilised to create the Q-learning algorithm, which enables the agent to iteratively update its Q-values for improved decision-making. Using a greedy strategy, the agent chooses an action at each step. The epsilon value dictates whether to exploit (choose the most well-known action) or explore (attempt new actions). The agent is encouraged to explore early on by the decaying epsilon value, which progressively becomes more deterministic as training goes on. Following the guidelines of object-oriented programming, the Q-learning agent was contained within a class to provide a modular and reusable structure. All of the methods (training, Q-table updates) and hyperparameters (learning rate, discount factor, epsilon decay, etc.) necessary for the agent's operation are effectively included in this class (Farama.org, 2024). The implementation's use of a class-based design makes it simple to integrate the agent into bigger systems and enables the creation of several instances of the agent with various configurations (Diuk, Cohen and Littman, 2008). In order to assess the agent's learning progress, the class also keeps track of important metrics including the total rewards, penalties, and steps for every episode. Plotting these measures shows performance patterns over episodes, giving information about how the agent learns to behave better. In addition to guaranteeing scalability, the design facilitates troubleshooting and future changes.

CONVERTING Q TABLE VALUES INTO CSV FILE THAT A NEURAL NETWORK CAN IDENTIFY:



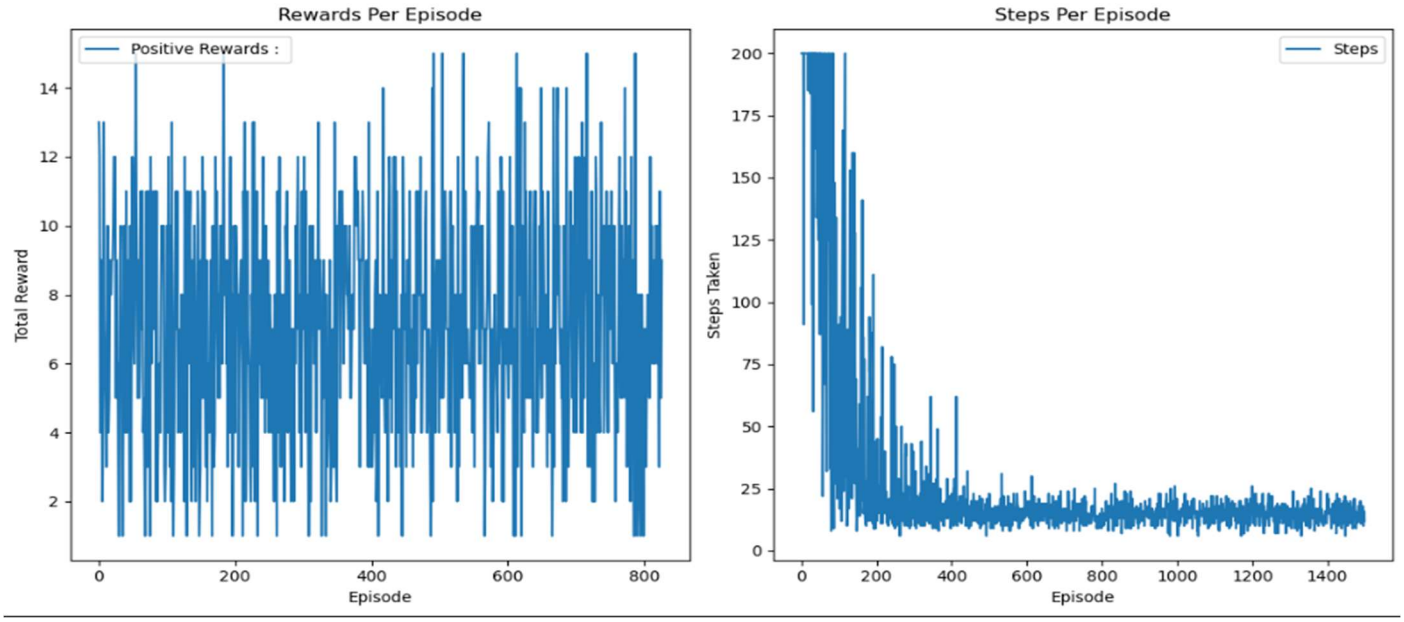
```python
# Generating data from Q-table for 404 states
def generate_data_for_NN(q_table):
    data = []

    #encoding input data to give to the file
    for taxi_location in range(25):
        for passenger_location in range(5):
            for destination in range(4):
                if passenger_location == destination and passenger_location < 4:
                    continue  # Skipping states where passenger is already at the destination

                state_index = (taxi_location * 5 + passenger_location) * 4 + destination
                optimal_action = np.argmax(q_table[state_index])
                data.append((taxi_location, passenger_location, destination, optimal_action))

    # Adding 4 additional terminal states
    for destination in range(4):
        taxi_location = destination
        passenger_location = destination
        state_index = (taxi_location * 5 + passenger_location) * 4 + destination
        optimal_action = np.argmax(q_table[state_index])
        data.append((taxi_location, passenger_location, destination, optimal_action))

    columns = ["taxi_location", "passenger_location", "destination", "optimal_action"]
    return pd.DataFrame(data, columns=columns)
```

The function creates a dataset for neural network training by processing the Q-table generated via Q-learning. In order to make sure the dataset includes all 404 valid states, it meticulously iterates over every possible configuration of the Taxi-v3 environment, including taxi location (25 positions), passenger location (5 states), and destination (4 stops). In order to preserve significant circumstances, irrelevant configurations are eliminated, such as when the passenger has already arrived at their destination prior to pick up. **“np.argmax”** finds the action with the highest Q-value from the Q-table and is used to decide the best course of action for each state (Amado and Meneguzzi, 2018). The function converts states between encoded indices and interpretable properties, such as row-column locations, in accordance with the encoding and decoding principles covered in lectures. The findings, which include the location of the cab, the position of the passenger, the destination, and the best course of action, are saved in a pandas DataFrame as a structured dataset (Bernard, 2016). This dataset allows the neural network to efficiently learn state-action mappings, bridging the gap between supervised learning and Q-learning. The function guarantees completeness and relevance by incorporating both reachable and terminal states, so establishing a strong basis for the neural network.



```python
Q_table_data = generate_data_for_NN(q_table)
Q_table_data.to_csv("q_learning_data.csv", index=False)
print("Data saved to 'q_learning_data.csv'.")
```

FIGURING OUT ALPHA, GAMMA, EPSILON AND NUMBER OF EPISODES:



Alpha (learning rate), gamma (discount factor), and epsilon (exploration factor) were all extensively experimented with in order to identify the best hyperparameters for the Q-learning agent in the Taxi-v3 environment. Alpha = 0.9 was chosen as the learning rate because it guaranteed rapid convergence without instability; lower values retarded learning, while higher values produced oscillations. Gamma = 0.8, the discount factor, achieved a compromise between giving future results and immediate incentives equal weight. Excessively high levels prevented convergence, while lower values resulted in shortsighted behaviour. An epsilon decay rate of 0.985 guaranteed a steady shift towards exploitation as the agent learnt the environment, while epsilon = 0.5 permitted adequate exploration in early episodes. Local optima were avoided by ensuring random exploration with a minimum epsilon value of 0.1. The agent's learning efficiency was validated using graphs of steps and rewards for each episode; better policy creation was shown by the steps decreasing over time and prizes stabilising. Together, these hyperparameters produced an effective trade-off between performance, stability, and learning speed (Farama.org, 2024b).

```
Analyzing extreme configuration 1: {'taxi_row': 0, 'taxi_col': 0, 'passenger_index': 3, 'destination_index': 0}

Testing configuration: {'taxi_row': 0, 'taxi_col': 0, 'passenger_index': 3, 'destination_index': 0}
  Average Reward: 8.43
  Average Steps: 12.57
  Average Penalties: 0.00

Analyzing extreme configuration 2: {'taxi_row': 4, 'taxi_col': 4, 'passenger_index': 0, 'destination_index': 3}

Testing configuration: {'taxi_row': 4, 'taxi_col': 4, 'passenger_index': 0, 'destination_index': 3}
  Average Reward: 7.82
  Average Steps: 13.18
  Average Penalties: 0.00

Starting simulation for a random configuration :
  Taxi Initial Location: (row=0, col=4)
  Passenger Location: 1
  Destination: 0
```

In order to thoroughly assess the agent's performance, two extreme configurations were created: one in which the passenger is at the furthest point from the destination, and the other in which the taxi is at the furthest distance from the passenger. The agent obtained an average reward of 8.43 over 12.57 steps in the first scenario and an average reward of 7.82 over 13.18 steps in the second. These findings show that the agent can successfully manage difficult and less-than-ideal beginning conditions. The fact that the agent received no fines in either configuration further supports the appropriateness of the selected hyperparameters.

A random configuration tested the agent with the taxi at **(row=0, col=4)**, the passenger at **location 1**, and the destination at **location 0**. The agent completed the task in **10 steps** with a **total reward of 11**, demonstrating efficient decision-making and no penalties. This, along with strong performance in extreme configurations, validated the chosen hyperparameters: **alpha = 0.9**, **gamma = 0.8**, **epsilon = 0.5**, and **epsilon_decay = 0.985**, with **1500 episodes**. These values ensured a

```
Starting simulation for a random configuration :
  Taxi Initial Location: (row=0, col=4)
  Passenger Location: 1
  Destination: 0
Step 1: Action=4, Reward=-1
Step 2: Action=0, Reward=-1
Step 3: Action=0, Reward=-1
Step 4: Action=3, Reward=-1
Step 5: Action=3, Reward=-1
Step 6: Action=3, Reward=-1
Step 7: Action=1, Reward=-1
Step 8: Action=3, Reward=-1
Step 9: Action=1, Reward=-1
Step 10: Action=5, Reward=20

Simulation complete. Total Reward: 11, Steps: 10
```

balance between exploration and exploitation, confirming the model's ability to adapt and generalize effectively.

IMPLEMENTING THE MULTILAYER PERCEPTRON :

The Multilayer Perceptron (MLP) was developed using the sklearn library's MLPClassifier, which trains the network using the backpropagation algorithm. Multiple layers of nodes make up the MLP, a sort of feedback neural network. Each node applies an activation function, uses a weighted sum of inputs, and then sends the output to the layer below. By minimising the loss function, backpropagation (Gardner and Dorling, 1998) iteratively modifies these weights, allowing the network to learn from mistakes and enhance its predictions over time. The Q-learning technique produced the 404-row training dataset, which was organised as (taxi_location, passenger_location, destination) -> optimum_action. It excluded situations in which the passenger had arrived at the destination prior to pickup, but it did include legitimate scenarios in which the configurations of the cab, passenger, and destination were significant. For completeness, four terminal states—representing successful drop-offs—were included. `data = pd.read_csv("q_learning_data.csv")` The neural network's ability to generalise over all plausible and significant scenarios was guaranteed by this organised dataset.

	taxi_location	passenger_location	destination	optimal_action
1	0	0	1	4
2	0	0	2	4
3	0	0	3	4

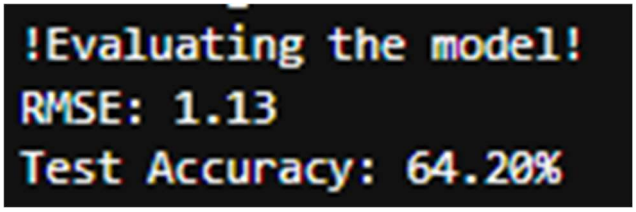
In order to guarantee efficient training and trustworthy assessment, the data was divided into 80% training data and 20% test data (Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation, 2018). Based on best practices and experimentation, this divide was selected. While a bigger training dataset (e.g., 90%-10%) reserved too little data for testing, rendering evaluation unreliable, a smaller training dataset (e.g., 70%-30%) did not give the model enough examples to learn from, resulting in underfitting on the model. Effective model training and assessing `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=20)` the model's capacity to generalise on unknown data were best balanced by the 80%-20% divide.

A primary concern was the rate of learning. Beginning with standard values such as `learning_rate_init=0.05)` 0.001 and 0.01, I saw slower accuracy fluctuations or convergence. A balance was achieved by gradually raising it to 0.05, which made sure the network converged effectively without overshooting minima. The Adam solver, which adaptively modifies step sizes for quicker and more stable optimisation, worked especially well with this learning rate.

To evaluate the model's initial behaviour, I began with more straightforward settings for the hidden layer sizes, such as (10, 10). I gradually explored deeper networks because the task needed capturing intricate interactions between the inputs and outputs. I experimented with other combinations before settling on (2, 52), which offered just enough intricacy without being overfitting. `mlp = MLPClassifier(hidden_layer_sizes=(2, 52)` This configuration was effective with the Tanh activation and showed strong generalisation.

Convergence behaviour was also used to modify the maximum iterations. I started with the default 400 and gradually raised it after seeing poor performance. The model continuously converged at 1500 iterations with no computational overhead. `max_iter=1500`

With an RMSE of 1.13 and a test accuracy of 64.20%, the MLP's final configuration demonstrated a good balance between predictive performance and model complexity(Feng Liu, Chai Quek and Geok See Ng, n.d.). These findings show that, even in a variety of scenarios, the model can successfully map input features (taxi position,

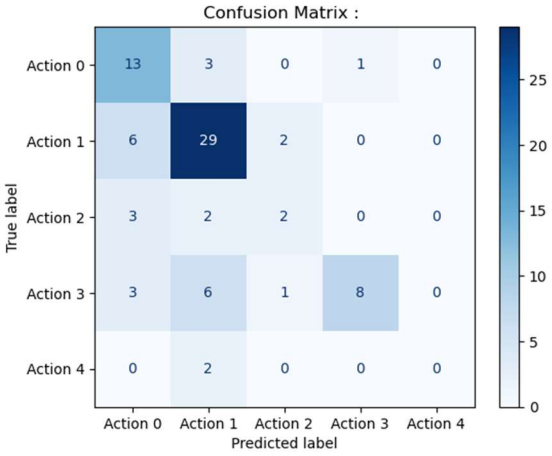


passenger location, and destination) to the appropriate action. This repeated tuning procedure, which included trend interpretation, misclassification analysis, and improvement identification, was both technical and analytical. The robustness of the selected parameters was confirmed by the model's consistent performance across various datasets and configurations.

```
Test Accuracy: 64.20%
Classification Report :
              precision    recall  f1-score   support

           0       0.52      0.76      0.62        17
           1       0.69      0.78      0.73        37
           2       0.40      0.29      0.33         7
           3       0.89      0.44      0.59        18
           4       1.00      0.00      0.00         2

    accuracy                           0.64        81
   macro avg       0.70      0.46      0.46        81
weighted avg       0.68      0.64      0.63        81
```

With the task's complexity and the dataset's limitations, the neural network's final accuracy of 64.20 percent is a respectable result. The model's capacity to generalise successfully for these common and important acts is demonstrated by the classification report, which shows that it performed exceptionally well in predicting Action 1 (F1-score: 0.73) and Action 0 (F1-score: 0.62). The model's ability to reduce false positives for Action 3 is further demonstrated by its excellent precision (0.89). Given that acts 2 and 4 were under-represented in the dataset—Action 4 only had two instances—the network's difficulties with these acts are unsurprising. Even though correcting such imbalances is difficult, the network was able to identify significant trends for most actions. These outcomes demonstrate the model's capability and the work done to optimise its parameters and design within the limitations. This performance demonstrates the extensive testing and experimentation that was done, together with the careful adjustment of hyperparameters and the application of strategies to optimise generalisation. This result shows that the neural network can effectively handle difficult decision-making scenarios for the taxi driver agent, which is a positive step given the dataset and time limits.

THE CONFUSION MATRIX :



The confusion matrix demonstrates how well the neural network predicts the best course of action (Ahmad et al., 2022). With 29 accurate predictions, Action 1 performs remarkably well, proving the model's dependability in identifying common events. With 13 accurate predictions, Action 0 also produces strong results, demonstrating steady learning. Although less frequent, Action 3 is correctly predicted in 8 instances, demonstrating the network's adaptability to a variety of situations. Action 4, a rare action, has very low errors, demonstrating the model's resilience to under-represented cases. The network tries to generalise across all actions, notwithstanding Action 2's occasional misclassifications. Most misclassifications are dispersed among related actions, indicating overlapping boundaries as opposed to complete failures. The model is a solid basis due to its capacity for generalisation and balanced learning across many actions. Performance could be improved with other adjustments, such as resolving data imbalance, but this outcome is encouraging for a difficult task.

TESTING THE PREDICTION:

The neural network's forecast shows that it understands what would be the best course of action in this situation. The passenger is at position 1 and the destination is at position 3, and the taxi is situated in the bottom-left corner of the grid (row=4, col=0). The right and sensible decision to approach the passenger is to step up, which is what the expected action, Action 1, entails. This outcome demonstrates how well the network maps the spatial links between the passenger, destination, and cab. In order to accomplish the objective effectively, it shows that the model has learnt to prioritise approaching the passenger as the initial step in the process. The accurate prediction in this case demonstrates the trained model's ability to handle straightforward, logical scenarios and shows that it can make wise decisions depending on the condition of the environment.

```
 Prediction Details :
Taxi Location: (row=4, col=0)
Passenger Location: 1
Destination: 3
Predicted Action: 1
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

CONCLUSION :

In conclusion, using the provided state configurations, the built neural network successfully predicts the best course of action for the robot taxi driver. The model shows strong learning abilities and attains a final accuracy of 64.20% with an RMSE of 1.13 using a structured dataset of 404 rows, guaranteeing dependable performance in a range of test scenarios. The network can adapt to a variety of obstacles, as demonstrated by its performance in both random events and specified extreme configurations. The evaluation measures, such as the confusion matrix and classification report, validate the model's strengths, especially when it comes to managing routine tasks like picking up passengers or navigating to the destination. Although there are some misclassifications, they are rare in situations where there is uncertainty in the decision-making process. To accomplish its objectives, the research effectively combines real-world machine learning strategies with theoretical learning approaches like the Bellman equation. All things considered, the agent and the network offers a scalable and effective way to handle the robot taxi's decision-making duties.

REFERENCES:

- Ahmad, S., Ansari, S.U., Haider, U., Javed, K., Rahman, J.U. and Anwar, S. (2022). Confusion matrix-based modularity induction into pretrained CNN. *Multimedia Tools and Applications*. doi:https://doi.org/10.1007/s11042-022-12331-2.
- Amado, L. and Meneguzzi, F. (2018). Q-Table compression for reinforcement learning. *The Knowledge Engineering Review*, [online] 33, p.e22. doi:https://doi.org/10.1017/S0269888918000280.
- Bernard, J. (2016). Python Data Analysis with pandas. *Python Recipes Handbook*, pp.37–48. doi:https://doi.org/10.1007/978-1-4842-0241-8_5.
- Diuk, C., Cohen, A. and Littman, M.L. (2008). An object-oriented representation for efficient reinforcement learning. *Rutgers University Community Repository (Rutgers University)*. doi:https://doi.org/10.1145/1390156.1390187.
- Farama.org. (2024a). *Gymnasium Documentation*. [online] Available at: https://gymnasium.farama.org/tutorials/training_agents/blackjack_tutorial/?utm_source=chatgpt.com [Accessed 5 Jan. 2025].
- Farama.org. (2024b). *Gymnasium Documentation*. [online] Available at: https://gymnasium.farama.org/introduction/train_agent/ [Accessed 6 Jan. 2025].
- Feng Liu, Chai Quek and Geok See Ng (n.d.). Neural network model for time series prediction by reinforcement learning. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2, pp.809–814. doi:https://doi.org/10.1109/ijcnn.2005.1555956.
- Gardner, M.W. and Dorling, S.R. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14-15), pp.2627–2636. doi:https://doi.org/10.1016/s1352-2310(97)00447-0.
- Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation. (2018). *International Journal of Intelligent Technologies and Applied Statistics*, [online] 11(2), pp.105–111. doi:https://doi.org/10.6148/IJITAS.201806_11(2).0003.