Tabular Q-Learning

Total Points: 45

Part 1: Tabular Q-Learning

Task 1: TaxiAgent Implementation (15 points)

Full marks for this task: 15 points

Item	Points	Criteria
Q-table initialization	3	Q-table is initialized as a 2D NumPy array with shape `(env.observation_space.n, env.action_space.n)` using the provided `q_init` parameter
Epsilon greedy action selection	3	`action_select()` method returns `env.action_space.sample()` with probability `epsilon`, otherwise returns `np.argmax(self.Q[s])`
Q-learning update - target calculation	3	`update()` method correctly computes target as `r` when `terminated` is True, and as `r + self.discount * np.max(self.Q[s_new])` when `terminated` is False
Q-learning update - Q-table update	3	`update()` method correctly applies the update rule: `self.Q[s_old, a] += self.lr * (target - self.Q[s_old, a])`
Epsilon decay	3	`update()` method decreases epsilon by multiplying by `epsilon_decay` while ensuring epsilon does not go below `epsilon_final` (using `max()` or equivalent logic)

Task 2: Training Implementation and Results (15 points)

Full marks for this task: 15 points

Item	Points	Criteria
Training loop structure	3	`train()` function includes: environment reset, action selection using `agent.action_select()`, environment step, and Q-learning update using `agent.update()` for each timestep
Episode termination handling	3	Training loop correctly handles episode termination by checking both `terminated` and `truncated` flags and resetting the environment when either is True
Training execution	3	Code successfully runs training for the specified number of episodes without errors
Learning curve visualization	3	Code generates both plots (episode returns and episode lengths) using the provided `visualize_learning_curves()` function
Convergence achievement	3	Final average return over the last 100 episodes is between -5 and +15 (indicating reasonable learning occurred)

Task 3: Reflection and Analysis Questions (15 points)

Full marks for this task: 15 points

Question 1 (5 points)

Item	Points	Criteria
Reason 1	2	Identifies Q-table initialization issue
Reason 2	2	Mentions high epsilon exploration
Reason 3	1	References invalid actions penalty

Question 2 (5 points)

Item	Points	Criteria
States convergence range	2	Provides a specific episode length convergence value or range
Explains optimal path reasoning	2	Explains that optimal pickup and delivery requires a certain minimum number of moves
Connects episode length to returns	1	Notes the relationship between shorter episodes and higher returns

Question 3 (5 points)

Item	Points	Criteria
Explains Q-value gradient	2	Describes how explored (s,a) pairs get updated
References action selection mechanism	2	Explains that `argmax` during exploitation will prefer unexplored actions
Mentions update rule specifics	1	References the Q-learning update equation and how the -1 reward propagates through the Q-table via the update rule

---

Optional Visualization Bonus

This is optional bonus credit for implementing the qualitative policy visualization

Item	Points	Criteria
Environment setup	1	Creates a new Taxi environment with render mode (either "human" or "ansi") and sets agent epsilon to 0 or a low value for evaluation
Evaluation loop structure	1	Implements a loop that runs multiple episodes (typically 3-5) with the trained agent, including environment reset, action selection, and episode termination handling

Results display	1	Displays or prints information about each episode such as the environment state (using `env.render()`), total reward, or steps taken

Deep Reinforcement Learning Assignment Rubric

Total Points: 55

Part 2: Deep Reinforcement Learning

Task 1: Neural Network Architecture (12 points)

Full marks for this task: 12 points

Item	Points	Criteria
Input layer dimension	3	Input layer (`fc1` or equivalent) accepts 8-dimensional input (first `nn.Linear` has input size 8)
Hidden layer structure	3	At least 2 fully connected hidden layers are defined with at least 64 neurons each
ReLU activation	3	ReLU (or equivalent non-linear) activation function is applied between layers (not on output layer)
Output layer dimension	3	Output layer produces 4-dimensional output (final `nn.Linear` has output size 4) with no activation function applied

---

Task 2: LunarLanderAgent Implementation (21 points)

Full marks for this task: 21 points

Item	Points	Criteria

Two network initialization	3	Both `q_network` and `target_network` are instantiated using the `NeuralNet` class and moved to the correct device
Target network parameter copying	3	Target network parameters are initialized by copying from the main Q-network using `load_state_dict()` or equivalent
Optimizer and loss initialization	3	An optimizer (e.g., Adam) is initialized with `q_network.parameters()` and a loss function (e.g., MSELoss) is defined
Replay buffer implementation	3	Replay buffer is implemented using `deque` with specified `maxlen`, and experiences (state, action, reward, next_state, terminated) are appended in `update()` method
Epsilon-greedy action selection	3	`action_select()` returns random action with probability epsilon, otherwise converts state to tensor, passes through `q_network`, and returns action with maximum Q-value using `argmax()`
Q-value computation	3	Current Q-values are computed by passing states through `q_network` and selecting values for taken actions using `gather()` or equivalent indexing
Target Q-value computation	3	Target Q-values are computed using `target_network` with `torch.no_grad()`, using the formula: `rewards + (1 - terminated) * discount * max(target_network(next_states))`

Task 3: Training and Results (12 points)

Full marks for this task: 12 points

Item	Points	Criteria
Training loop execution	3	Training code runs for at least 500 episodes without errors and includes action selection, environment step, and agent update
Terminated vs truncated handling	3	Only `terminated` flag (not `truncated`) is passed to `agent.update()` as the done signal

Environment interaction structure	3	Training loop includes: environment reset at start of each episode, action selection using `agent.action_select()`, environment step with `env.step()`, and agent update using `agent.update()` for each timestep
Learning outcome	3	The average/expected episode return converges to greater than 150

Task 4: Reflection and Analysis Questions (10 points)

Full marks for this task: 10 points

Question 1 (5 points)

Item	Points	Criteria
Argument 1	2	Explains short initial episodes
Argument 2	2	Explains long mid-training episodes
Argument 3	1	Explains short final episodes

Question 2 (5 points)

Item	Points	Criteria
Identifies state space	2	States the nature of the state space of Lunar Lander
Explains tabular limitation	2	Explains that tabular Q-learning requires discrete states
Explains DQN advantage	1	Notes that DQN uses a neural network to approximate the Q-function

Optional Qualitative Visualization Bonus

This is optional bonus credit for implementing the qualitative policy visualization

Item	Points	Criteria
Environment setup for visualization	1	Creates a new LunarLander environment with `render_mode="human"` or `render_mode="rgb_array"` and sets agent epsilon to 0 or a low value (≤ 0.1) for evaluation
Evaluation loop implementation	1	Implements a loop that runs multiple episodes (3-10 episodes) with the trained agent, including: environment reset, action selection using the trained agent, environment step, and episode termination handling
Proper evaluation mode	1	Does NOT call `agent.update()` during visualization (i.e., no learning updates during evaluation) and uses greedy or near-greedy action selection

---