Comparison Report: Tabular Q-Learning vs Deep Q-Learning (DQN) in Snake Game

1. Overview

This report compares two reinforcement learning strategies **Tabular Q-Learning** and **Deep Q-Learning (DQN)** applied to the classic **Snake game**, built using Pygame. Both agents were trained to maximize their performance by collecting food and avoiding collisions. The project aimed to understand how each method performs in a discrete grid environment under the same reward dynamics.

2. Task Description

The Snake game is a grid-based environment where an agent (the snake) learns to collect food placed randomly on the board while avoiding collisions with the walls or its own body. The agent receives rewards based on the outcome of each action:

- A reward of +10 when the snake eats food.
- A penalty of -10 when the snake collides with a wall or itself.
- A reward of 0 for any other movement.

The snake is allowed to take one of three possible actions at each step: move straight, turn left, or turn right.

3. Implementation Details

Tabular Q-Learning

- The state was represented using an 11-element binary feature vector. This vector encoded danger in three directions (ahead, left, right), the snake's current direction (up, down, left, right), and the food's position relative to the snake's head (left, right, above, below).
- A Python `defaultdict` was used to create the Q-table, mapping each discrete state-action pair to a Q-value.

- The agent followed an epsilon-greedy policy, which allowed it to balance between exploration and exploitation. Epsilon was gradually decayed with each episode.
- The Q-values were updated using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_a Q(s',a) - Q(s,a) \right]$$

- The reward system was as described in the task.

Deep Q-Learning (DQN)

- A neural network replaced the Q-table as a function approximator for Q-values.
- The network architecture included three fully connected layers with ReLU activations. The input layer accepted the 11-element state vector, and the output layer produced Q-values for each of the three possible actions.
- A replay buffer was used to store past experiences (state, action, reward, next state, done) and sample mini-batches during training to reduce correlation between samples.
- A separate target network was used to stabilize training by providing consistent targets for temporal-difference updates. It was updated every 10 episodes.
- The same reward system and state representation were used as in the tabular method.
- The neural network was trained using mean squared error loss and the Adam optimizer.

4. Training Setup

Both Q-learning and DQN agents were trained over 500 episodes. The grid size used was 20 by 20 cells. The initial epsilon value for exploration was set to 1.0 and decayed by a factor of 0.995 after each episode, with a lower bound of 0.01. The learning rate for Q-learning was 0.1, while the neural network used a learning rate of 0.001. The discount factor (γ) was set to 0.9 in both methods. The batch size for DQN was 64, and experiences were stored in a buffer of capacity 10,000. The DQN's target network was updated every 10 episodes. Actions available to the agent included moving straight, turning left, and turning right.

5. Results and Observations

Tabular Q-Learning

- The agent initially performed random movements and achieved low scores.
- Over time, it learned to associate certain state-action pairs with favorable outcomes.

- Its performance remained unstable due to the limitations of Q-tables in generalizing across similar states.
- The agent's performance varied significantly across episodes, with some achieving high scores while others failed early.

Deep Q-Learning

- The initial behavior was similar, but the agent began to learn optimal strategies earlier in the training.
- The learning curve was smoother and more stable.
- The neural network was better able to generalize across similar but not identical states.
- The DQN agent frequently achieved scores over 200 in the later episodes and exhibited more consistent behavior in gameplay.

6. Analysis and Comparison

Several key differences were noted:

- **Learning Speed**: The DQN agent began improving earlier in the training process, while Q-learning took longer to show progress.
- **Stability**: DQN performance improved more smoothly over time, whereas tabular Q-learning exhibited significant fluctuations.
- **Generalization**: DQN generalized better across unseen states, while tabular Q-learning was limited to previously encountered exact states.
- **Scalability**: Tabular Q-learning was limited due to state explosion, while DQN managed complexity effectively through function approximation.
- **Final Scores**: DQN agents consistently achieved higher and more stable scores than tabular agents.

7. Conclusion

This project demonstrated the strengths and weaknesses of classical and modern reinforcement learning methods in a dynamic environment like the Snake game. While tabular Q-learning serves as a foundational method suitable for small, finite state spaces, it struggles to scale or generalize in more complex settings.

Deep Q-Learning, on the other hand, leverages neural networks and experience replay to overcome these limitations. It offers better convergence speed, performance consistency, and adaptability to unseen situations.

In conclusion, Deep Q-Learning proved to be the superior approach for solving the Snake game due to its ability to generalize, learn faster, and perform more robustly in a complex and growing environment.