Comparative Analysis of Q-Learning and Deep Q-Network (DQN) in the Snake Game

1. Overview

This report presents a comparative evaluation of two reinforcement learning strategies—Q-Learning and Deep Q-Network (DQN)—applied to the classic Snake game using the Pygame framework. The objective is to train agents that learn to play the game effectively by maximizing cumulative rewards through exploration and experience.

2. Game Description

The Snake game operates on a discrete grid where the snake's objective is to consume randomly placed food while avoiding collisions with the boundaries or its own body. The game provides positive reinforcement when the snake eats food and negative feedback when a collision occurs. The state of the game is encoded based on directional hazards, current movement direction, and relative position of the food, which serves as input to both learning models.

3. Q-Learning Implementation

The Q-Learning approach is implemented using a tabular method. The game state is transformed into a binary tuple that captures the presence of danger in forward, left, and right directions, current movement direction (left, right, up, down), and the relative position of the food. This results in a discrete state space manageable through a Q-table.

Actions are encoded as integers representing movement: 0 (forward), 1 (right turn), and 2 (left turn). The algorithm follows an epsilon-greedy strategy for action selection to balance exploration and exploitation. The Q-table is updated using the Bellman equation with learning rate ($\alpha = 0.1$), discount factor ($\gamma = 0.9$), and an exploration decay factor of 0.995 per episode until a minimum threshold.

Rewards are assigned as follows: +10 for eating food, 10 for crashing into a wall or itself, and 0 for other movements. The agent was trained over 500 episodes, and total reward was tracked and visualized.

4. Deep Q-Network (DQN) Implementation

The Deep Q-Learning model replaces the Q-table with a neural network that approximates Q-values for each possible action. The state input is the same binary tuple used in the tabular method. The neural network consists of three fully connected layers with ReLU activation functions, transforming an 11-dimensional input into a 3-dimensional output corresponding to action scores.

To enhance learning stability, the model employs experience replay using a buffer and periodic updates of a target network. The loss function used is Mean Squared Error (MSE) between predicted Q-values and target values. The agent is optimized using the Adam optimizer with a learning rate of 0.001.

A batch size of 64 is used for training from the replay buffer. Exploration is controlled via an epsilon value that decays after each episode, starting from 1.0 and gradually reducing to 0.01.

5. Training Configuration

Both models were trained for 500 episodes with a discount factor of 0.9. The Q-learning agent used a fixed learning rate and decaying epsilon. The DQN agent included additional enhancements like a target network updated every 10 episodes and a replay buffer with a capacity of 50,000 experiences.

6. Results and Performance

The Q-Learning agent exhibited noticeable improvement over time, with an increase in average total reward per episode. The reward graph shows fluctuations due to the discrete nature of the state space and occasional instability. The highest rewards hovered around 200, but there were frequent low-reward episodes caused by overfitting to seen states.

In contrast, the DQN agent demonstrated more consistent performance. The learning curve was smoother and more stable, and rewards steadily increased throughout training. The DQN agent showed superior generalization capabilities, with scores often exceeding those of the tabular agent in later episodes.

7. Observations and Insights

The following key insights were observed from the training process:

- **Stability**: The DQN model exhibited greater stability in reward progression, whereas the Q-learning agent showed higher volatility.
- **Generalization**: DQN generalized better to unseen scenarios due to function approximation, while Q-learning struggled with rare or new states.
- **Performance Peaks**: While the Q-learning agent achieved occasional high spikes in rewards, the DQN agent maintained higher average performance consistently.

- **Learning Efficiency**: Deep Q-Learning required more computational resources but offered improved learning efficiency over time.

8. Comparison of Q-Learning and DQN Approaches

The following points summarize the key differences observed between the two approaches during training and evaluation:

Q-Learning (Tabular)

- Works well with a small and fully observable discrete state space.
- Simple to implement and computationally efficient.
- Performance is highly dependent on how well the state is discretized.
- Exhibits sharp fluctuations in learning, especially when encountering unseen states.
- Learning is limited to exact state-action pairs and does not generalize well.
- Achieved occasional high rewards but lacked consistency across episodes.

Deep Q-Network (DQN)

- Capable of handling larger or continuous state spaces using function approximation.
- Generalizes across similar states, enabling more robust decision-making.
- More computationally intensive and requires additional components such as replay buffer and target network.
- Demonstrated smoother and more stable learning curves throughout training.
- Required more episodes to converge but produced consistently higher average rewards in the long run.
- Better suited for dynamic or expanding environments like Snake, where the state space grows as the snake increases in length.

9. Conclusion

This experiment highlights the trade-offs between classic tabular methods and deep learning-based function approximation. For tasks with large or continuous state spaces like the Snake game, DQN offers significant advantages in scalability, learning stability, and long-term performance. While Q-learning remains a viable approach for simpler environments, DQN is better suited for more complex decision-making tasks such as dynamic, growing gameplay environments.