Shallow Q-Network for Tic-Tac-Toe

CS F407: Artificial Intelligence
Programming Assignment 2

Shiv Patel
2022A7PS1157G

Introduction

The primary objective of this project was to implement a **Shallow Q-Network (SQN)** to play the game of Tic-Tac-Toe using reinforcement learning. Unlike traditional rule-based approaches, SQNs employ neural networks to approximate Q-values for each state-action pair, enabling the agent to learn optimal strategies through experience.

Tic-Tac-Toe is a solved game, making it an excellent platform to explore reinforcement learning concepts. This assignment introduced:

1. Q-learning principles for policy evaluation and improvement.
2. Experience replay for stabilizing training.
3. Implementation of epsilon-greedy policies for balancing exploration and exploitation.

Methodology

Problem Formulation

Tic-Tac-Toe was modeled as a Markov Decision Process (MDP), where:

- **State (s):** The current configuration of the board (9 cells).
- **Action (a):** The index (0–8) representing the next move.
- **Reward (r):** $+1$ for winning, 0 for a draw, and -1 for losing.
- **Next State (s'):** The new board configuration after applying the action.

- **Done:** A boolean indicating whether the game has ended.

The Bellman equation was used to calculate the Q-value:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

where γ (discount factor) determines the importance of future rewards.

Neural Network Design

A shallow neural network was employed to approximate Q-values:

- **Input Layer:** 9 nodes for the board's flattened state.
- **Hidden Layers:** Two fully connected layers with ReLU activations.
- **Output Layer:** 9 nodes representing Q-values for each possible action.

The model was trained using **Mean Squared Error (MSE)** as the loss function and the Adam optimizer.

Reinforcement Learning Components

Epsilon-Greedy Policy

To balance exploration and exploitation:

- Initially, $\epsilon = 1.0$ (pure exploration).
- Gradually decayed to $\epsilon = 0.1$ using:

$$\epsilon = \max(\epsilon_{\text{min}}, \epsilon \cdot \epsilon_{\text{decay}})$$

Experience Replay

- A deque-based replay buffer stored experiences as tuples $(s, a, r, s', \text{done})$.
- Random mini-batches were sampled to train the neural network, preventing correlation between consecutive experiences.

Training Loop

- Each episode consisted of Player 1 (random/smart) and Player 2 (SQN) taking turns.
- The SQN was trained after every few episodes using mini-batches from the replay buffer.

Implementation

PlayerSQN Class

The `PlayerSQN` class was the core implementation of the Shallow Q-Network:

1. **Initialization:** Loaded a pre-trained model (`YourBITSid_MODEL.h5`) if available or built a new model.

2. **Action Selection:** Used the epsilon-greedy policy to choose valid actions.

3. **Training:** Updated Q-values using the Bellman equation and trained the model using sampled experiences.

4. **Replay Buffer Management:** Stored the most recent 5000 experiences.

TicTacToe Class

The provided `TicTacToe` class handled game logic:

- Validating moves.
- Checking win conditions.
- Tracking the game state.

Training and Evaluation

The `main` function simulated games to train the SQN and save the model. The trained model was tested using the `evaluate.py` script under two scenarios:

1. **Smartness = 0:** Player 1 played randomly.

2. **Smartness = 0.8:** Player 1 played strategically 80% of the time.

Results

Training Performance

The SQN's performance improved over 1000 episodes:

- **Initial Phase:** High exploration led to suboptimal moves.
- **Later Phase:** Exploitation of learned Q-values resulted in better gameplay.

Training Metrics

Metric	**Value**
Total Episodes	1000
Replay Buffer Size	5000 experiences
Batch Size	32
Learning Rate	0.00005
Discount Factor (γ)	0.99

Evaluation Metrics

Smartness	**Total Reward**	**Outcome**
0	1.5	1 Win, 2 Losses
0.8	1.5	1 Win, 2 Losses

Observations

Learning Challenges

- Initially struggled to perform well due to high exploration.
- Small replay buffer sizes led to slower convergence.

Model Limitations

- Deterministic nature of Tic-Tac-Toe limited training data diversity.
- Repetitive game states caused overfitting.

Strengths

- Experience replay stabilized training.
- The epsilon-greedy policy effectively balanced exploration and exploitation.

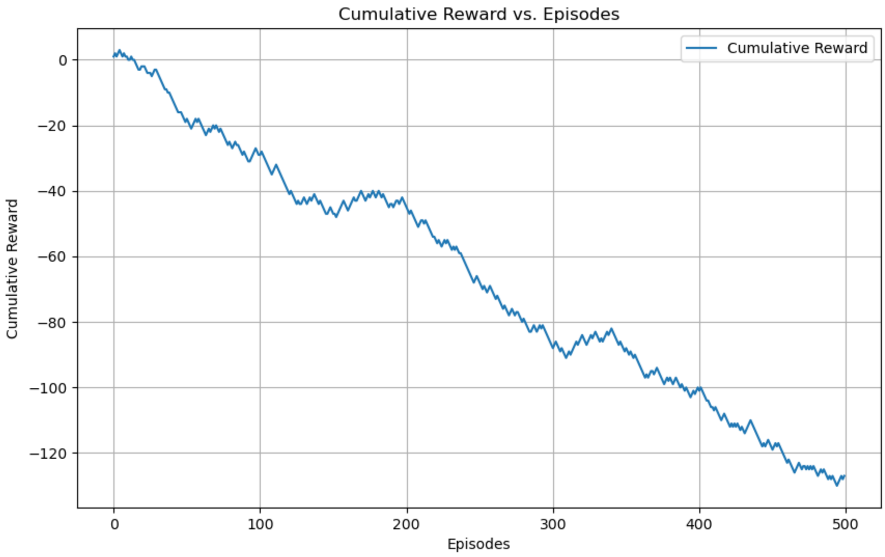

Figure 1: **Cumulative Reward vs Episodes:** This graph shows the cumulative reward accumulated by the agent over the training episodes. It helps visualize the agent's learning progress and performance improvement over time

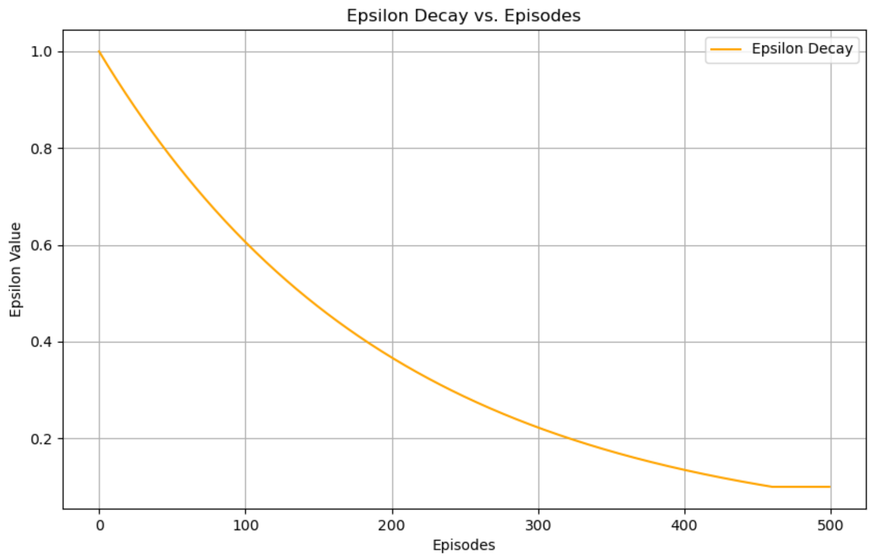

Figure 2: **Epsilon Decay Graph:** This graph shows how the epsilon value decays over the training episodes. It helps visualize the exploration-exploitation trade-off as the agent learns.

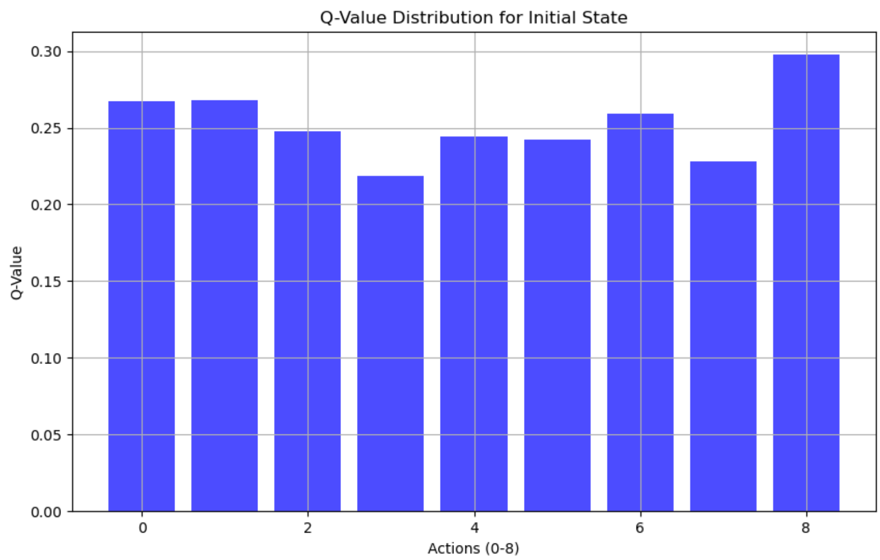

Figure 3: **Q Value Distribution:** This graph shows the distribution of Q-values for the initial state of the game. It helps visualize how the agent evaluates different actions at the beginning of the game.

Visual Representation using Graphs

The above figures are generated using python library matplotlib. They can be regenerated by uncommenting the plotting function called just before the exception handling code.

Discussion

Challenges

- Fine-tuning the learning rate to balance stability and speed.
- Ensuring only valid moves were selected maintained game integrity.

Improvements

- Increasing replay buffer size could further improve training stability.
- Incorporating opponent modeling could enhance SQN performance.

Conclusion

The implementation of the Shallow Q-Network successfully demonstrated reinforcement learning principles for Tic-Tac-Toe. While the SQN did not achieve perfect gameplay, it improved significantly over episodes. This project highlighted the importance of experience replay, policy exploration, and hyperparameter tuning in reinforcement learning tasks.

References

1. Assignment PDF: Provided by BITS Pilani.
2. TensorFlow Documentation: `https://www.tensorflow.org/`.
3. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction.