Open-Ended Lab Report: Solving Breakout Using Reinforcement Learning

Tayyib ul Hassan

December 22, 2024

Introduction

This report details my work on the open-ended lab assignment ”Solving Breakout Using Reinforcement Learning,” as part of the CS368: Reinforcement Learning course. The primary objective was to apply reinforcement learning (RL) algorithms to solve the Breakout Atari game from the Gymnasium library. I implemented a Deep Q-Network (DQN) for this purpose, optimized its performance, and analyzed the results using graphs generated during training.

The Breakout game, a classic Atari game, requires controlling a paddle to hit a ball and break bricks. It presents significant challenges for RL algorithms, including precise control and strategic planning, due to its dynamic gameplay and sparse rewards.

Objectives

The main objectives of this lab were as follows:

- Develop, train, and evaluate a reinforcement learning agent to solve the Breakout Atari game.
- Preprocess the environment and optimize the agent’s performance.
- Analyze the agent’s performance using graphs and metrics.

Tools and Libraries

I used the following tools and libraries for this lab:

- **Python**: For scripting and implementing the RL algorithm.
- **Gymnasium**: For interacting with the Atari Breakout environment.
- **PyTorch**: For building and training the Deep Q-Network.
- **Optuna**: For optimizing hyperparameters.
- **Matplotlib**: For plotting and visualizing metrics.
- **OpenCV**: For preprocessing game frames.

Implementation Details

Deep Q-Network (DQN)

I implemented a Deep Q-Network to approximate the Q-values for different actions. The architecture includes convolutional layers to process the visual input (game frames) and fully connected layers for decision-making.

Frame Preprocessing

Each game frame was converted to grayscale, resized to 84x84 pixels, and normalized to improve training efficiency. Four consecutive frames were stacked to capture temporal dependencies.

Replay Buffer

I used an experience replay buffer to store transitions (state, action, reward, next state, done) and sample batches for training. This helped stabilize learning by breaking correlations between consecutive transitions.

Epsilon-Greedy Policy

An epsilon-greedy policy was employed to balance exploration and exploitation. The epsilon value decayed over time to encourage exploration in the early stages and exploitation in the later stages.

Optimized Hyperparameters

I optimized the hyperparameters of the DQN using Optuna. The optimized values are as follows:

- **Gamma (Discount Factor)**: 0.85
- **Learning Rate**: 1×10^{-4}
- **Batch Size**: 32
- **Replay Buffer Size**: 10,000
- **Minimum Replay Buffer Size**: 1,000
- **Epsilon Start**: 0.5
- **Epsilon End**: 0.1
- **Epsilon Decay**: 1,000,000
- **Target Network Update Frequency**: 100 steps
- **Maximum Episodes**: 500
- **Maximum Steps per Episode**: 1,000

Algorithm: Deep Q-Network (DQN)

Algorithm 1 Deep Q-Network (DQN) for Breakout

```
1: Initialize replay buffer $\mathcal{D}$ to capacity $N$
2: Initialize action-value function $Q(s, a; \theta)$ with random weights $\theta$
3: Initialize target action-value function $Q^{\text{target}}(s, a; \theta^-)$ with weights $\theta^- = \theta$
4: **for** episode = 1 to Maximum Episodes **do**
5:     Initialize state $s_0$
6:     **for** timestep = 1 to Maximum Steps per Episode **do**
7:         Select action $a_t$ with epsilon-greedy policy:
8:             $a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a; \theta) & \text{otherwise} \end{cases}$
9:         Execute action $a_t$ in environment
10:        Observe reward $r_t$ and next state $s_{t+1}$
11:        Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in $\mathcal{D}$
12:        Sample random mini-batch of transitions $(s_j, a_j, r_j, s_{j+1}, \text{done}_j)$ from $\mathcal{D}$
13:        Compute target for each transition:
14:            $y_j = \begin{cases} r_j & \text{if done}_j = \text{true} \\ r_j + \gamma \max_{a'} Q^{\text{target}}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
15:        Update weights $\theta$ by minimizing the loss:
16:            $\mathcal{L}(\theta) = \frac{1}{|\text{mini-batch}|} \sum_j \left(y_j - Q(s_j, a_j; \theta)\right)^2$
17:        Every $C$ steps, update target network weights:
18:            $\theta^- \leftarrow \theta$
19:        **if** done **then**
20:            Break from loop
21:        **end if**
22:    **end for**
23: **end for**
```

Results and Analysis

The training process was tracked using various metrics:

- **Total Reward per Episode**: The total rewards graph provides insights into the agent's performance improvements throughout training. In the early episodes, the rewards were relatively low, as the agent lacked prior knowledge about the environment. As training progressed, the rewards exhibited an upward trend, signaling the agent's ability to learn effective strategies for hitting the ball and breaking bricks. The variability in rewards, especially during the intermediate episodes, highlights the exploration phase where the agent was still experimenting with different actions. The eventual increase in rewards demonstrates successful learning and policy refinement.

- **Epsilon Decay**: The epsilon decay graph illustrates the gradual reduction in the exploration factor (epsilon) over the course of training. Initially, the epsilon value was set at 0.5 to encourage exploration, allowing the agent to sample diverse actions and learn from various states. As training progressed, epsilon decreased linearly toward 0.1, emphasizing exploitation of the learned policy. This decay strategy is crucial for transitioning from exploration (early learning phase) to exploitation (refinement phase). The smooth reduction observed in the graph indicates a stable balance between exploration and exploitation during training.

- **Loss per Episode**: The loss graph captures the learning dynamics of the DQN. During the initial episodes, the loss values were relatively high, reflecting the agent's efforts to approximate the Q-values from scratch. Over time, the loss exhibited fluctuations, indicative of the stochastic nature of reinforcement learning and the continuous updates to the Q-function. However, the overall trend demonstrates convergence, as the loss starts stabilizing toward later episodes. Periodic spikes in the graph can be attributed to challenging transitions or significant updates to the target network.

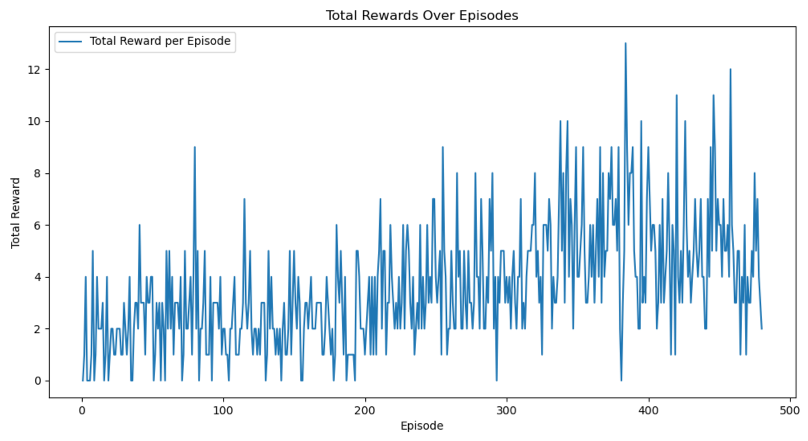

Figure 1: Total Rewards Over Episodes

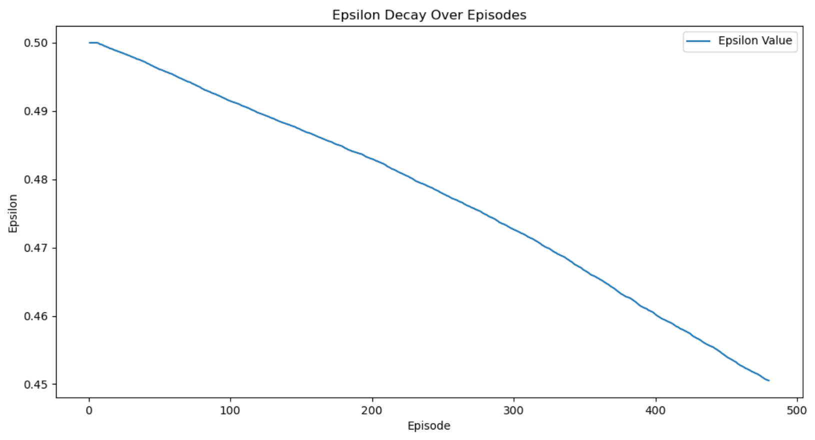

Figure 2: Epsilon Decay Over Episodes

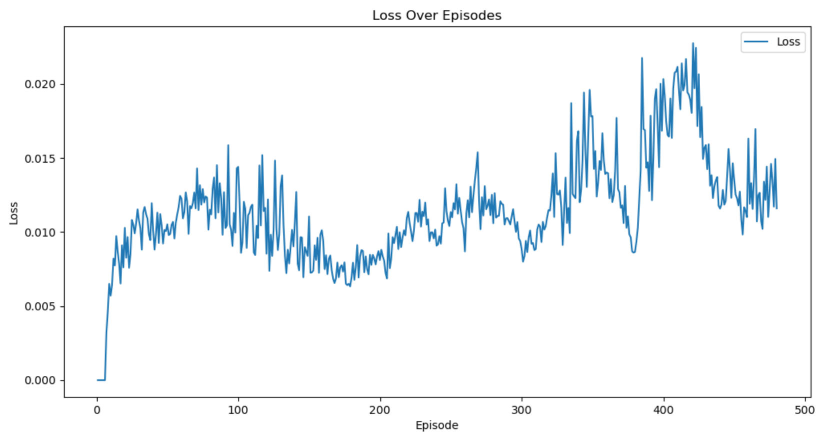

Figure 3: Loss Over Episodes

Conclusion

In this lab, I successfully implemented and trained a Deep Q-Network to solve the Breakout Atari game. The optimized hyperparameters, determined through Optuna, significantly improved the agent's performance. The agent demonstrated a clear learning curve, as evidenced by the increasing rewards, decreasing loss, and successful epsilon decay.

The lab provided hands-on experience with RL algorithms, frame preprocessing, and performance optimization. I gained a deeper understanding of the challenges in RL and how to address them through systematic experimentation and hyperparameter tuning.

References

- Gymnasium Documentation: `https://gymnasium.farama.org/`
- PyTorch Documentation: `https://pytorch.org/`
- Optuna Documentation: `https://optuna.org/`