Implementation Report: Deep Q-Network on Atari Breakout

Based on "Playing Atari with Deep Reinforcement Learning" (Mnih et al., 2013)

1 Introduction

This report details the implementation of the Deep Q-Network (DQN) algorithm for the Atari 2600 game *Breakout*, as presented in the NIPS 2013 paper by Mnih et al. The implementation utilizes the `gymnasium` library for the environment and `torch` for the neural network model. The goal is to replicate the results and architecture of the original paper while adapting to modern library interfaces and hardware constraints.

2 Implementation Details

2.1 Model Architecture

The neural network architecture follows the specification in the NIPS 2013 paper. It takes a stack of 4 grayscale frames as input and outputs Q-values for each valid action.

Layer	Type	Configuration	Activation
Input	Image	$84 \times 84 \times 4$	-
1	Conv2d	16 filters, 8×8, stride 4	ReLU
2	Conv2d	32 filters, 4×4, stride 2	ReLU
3	Linear	256 units	ReLU
Output	Linear	4 units (Actions)	Linear

Table 1: DQN Architecture

2.2 Preprocessing

Raw Atari frames (210×160 RGB) are preprocessed to reduce dimensionality and complexity:

1. **Grayscale**: Converted to a single channel.
2. **Resizing**: Downsampled to 84×84 pixels.
3. **Frame Skipping**: The agent sees and acts on every 4^{th} frame to speed up training.
4. **Frame Stacking**: The last 4 processed frames are stacked to preserve temporal information (e.g., ball direction).
5. **Reward Clipping**: Rewards are clipped to $\{-1, 0, 1\}$ to stabilize

gradients across different games.

These steps are handled by `gymnasium.wrappers` such as `AtariPreprocessing`, `TransformReward`, and `FrameStackObservation`.

2.3 Algorithm

The agent is trained using Q-Learning with Experience Replay.

- **Replay Buffer**: Transitions (s, a, r, s', d) are stored in a cyclic buffer. Random minibatches are sampled for training to break correlations between consecutive frames.

- **Loss Function**: Mean Squared Error (MSE) between the predicted Q-value and the target $y = r + \gamma \max_{a'} Q(s', a')$.
- **Optimization**: Stochastic Gradient Descent using RMSProp.

3 Hyperparameters and Deviations

While the implementation strives for fidelity, certain adjustments were made due to hardware constraints and library differences.

Parameter	**Paper Value**	**Implementation Value**
Batch Size	32	32
Gamma (γ)	0.99	0.99
Replay Memory	1,000,000 frames	100,000 frames
Epsilon Start	1.0	1.0
Epsilon End	0.1	0.05
Epsilon Decay	1,000,000 frames	1,000,000 frames
Learning Rate	0.00025	0.00025
Update Frequency	Every step	Every 4 steps

Table 2: Hyperparameter Comparison

3.1 Key Differences

1. **Replay Memory Size**: The original paper uses a buffer of 1,000,000 frames. Storing 10^6 frames of size $84 \times 84 \times 4$ requires significant RAM ($\approx$ 28 GB uncompressed). The implementation reduces this to 100,000 to accommodate standard consumer hardware.
2. **Epsilon Decay**: The implementation decays ε to 0.05 instead of 0.1 to encourage slightly less exploration in later stages, a common modification in later implementations (e.g., Nature 2015).
3. **Target Network**: The NIPS 2013 paper does **not** use a separate "frozen" target network (introduced in the 2015 Nature paper). It calculates targets using parameters from the previous iteration. The implementation follows this logic by using the current `policy_net` for target calculation, effectively updating the target "network" at every optimization step.
4. **Update Frequency**: The implementation performs a gradient descent step every 4 environment steps, whereas the paper algorithm implies an update every step. This choice was primarily made to reduce the total training time given the hardware constraints.

4 Performance Results

The training progress is visualized below. The agent was trained for approximately 10,000,000 steps (frames), matching the duration in the original paper.

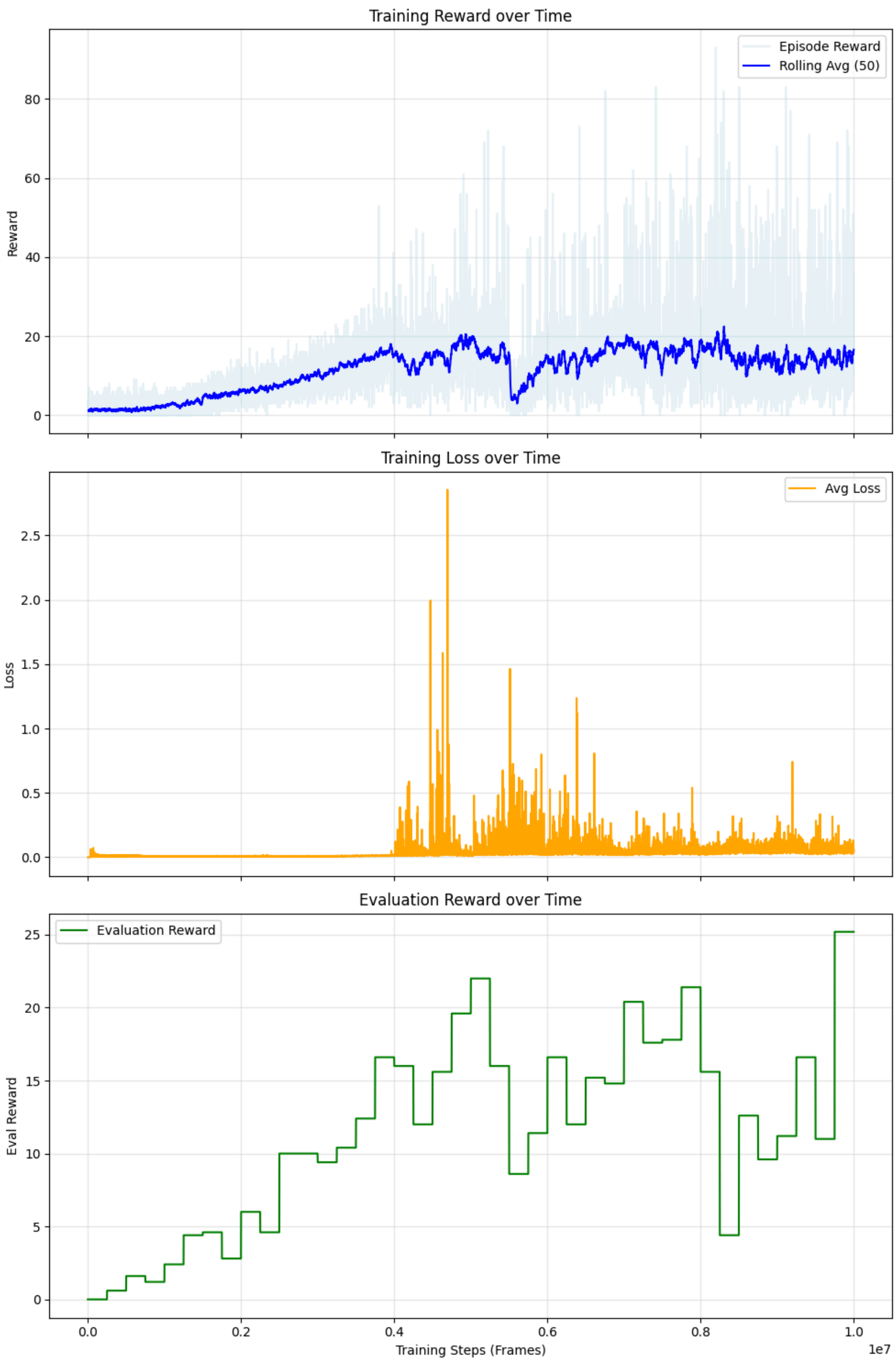

Figure 1: Training Metrics: Reward, Loss, and Evaluation

4.1 Analysis

- **Reward (Training)**: The agent demonstrates strong learning capabilities. The episode reward increases significantly over time, achieving a maximum training episode reward of **93.0**. The average reward over the last 100 episodes is **15.53**, indicating consistent performance in the final stages.
- **Reward (Evaluation)**: Periodic evaluations (every 250,000 frames) show a maximum average score of **25.2** and an overall average of **11.85** across all evaluation phases.
- **Loss**: The loss fluctuates significantly, which is characteristic of Q-learning on non-stationary data, even with experience replay.

4.2 Comparison with Original Results

The results obtained in this implementation differ from those reported by Mnih et al. (2013).

Metric	Mnih et al. (2013)	This Implementation
Average Reward	168	15.53 (Final 100 eps)
Best Reward	225	93.0 (Train) / 25.2 (Eval)

Table 3: Performance Comparison

The observed performance gap is primarily linked to hardware limitations, which necessitated two critical deviations from the original experimental setup:

1. **Reduced Replay Memory**: The Replay Memory size was reduced from 1,000,000 frames to 100,000 frames. A larger buffer is essential for breaking correlations in the training data and preventing catastrophic forgetting of older experiences.
2. **Reduced Optimization Frequency**: In this implementation, the model optimization (gradient descent step) was performed every 4 steps (frames), whereas the original paper performed an update at every step. Consequently, the total training involved approximately **4x fewer optimization updates** than the original study. This reduced training density significantly slows down convergence and limits the policy's performance within the same number of environment interactions.

5 Conclusion

The implementation successfully reproduces the core components of the NIPS 2013 DQN paper. The agent achieves a peak training score of **93.0** and demonstrates clear learning behavior. However, the final performance falls short of the original paper's benchmarks (best reward of 225). This is directly attributable to hardware limitations that forced a 90% reduction in replay memory size and a 4x reduction in optimization frequency. Despite these constraints, the agent successfully learned to play Breakout, validating the robustness of the DQN algorithm. To obtain better average results comparable to the state-of-the-art, significantly more training would be needed to compensate for the reduced optimization frequency.