CSC 580 AI II | HW5 | NAME: OM PRAKASH GUNJA | STUDENT ID: 2131025

Deep Q-Learning for Atari Breakout

Introduction

The objective was to analyze and optimize the decision-making process of an agent using reinforcement learning, aiming to enhance its performance over time through hyper parameter tuning.

The key focus was on systematically improving training efficiency, reward maximization, and model stability. To achieve this, I implemented various approaches.

- Prioritized Experience Replay (PER) for sampling efficiency.
- Dynamic batch sizing based on loss variations.
- Gradient clipping & loss stabilization techniques.
- Frequent target network updates for improved convergence.
- Adaptive exploration-exploitation strategies.

Evolution of Models

1. Base Model(DQN Basic) Estimated run time 35+ hrs on CPU

Initial implementation with:

- Basic DQN implementation
- Simple replay memory using deque
- Fixed batch size
- Basic preprocessing using PIL for image resizing
- Simple training loop

The model is extremely slow and requires significant enhancements. The estimated time for complete execution is over 35 hours. Therefore, I decided to focus on improvements, including PER and other enhancements. Here's a progress report on these enhancements and their performance:

2. PER Version (580_hw5_PER.ipynb) Ran for 10+ hrs on CPU

In this version, I have made several updates after thorough research.

Some of them inclose

2.1. Image Processing

- Added OpenCV (cv2) based preprocessing
- Kept original PIL version for compatibility
- More efficient frame cropping and resizing

```
import cv2
def crop_Atari_frame_cv2(frame):
        # Define cropping region
    x_start = 5  # Starting x-coordinate
    y_start = 30  # Starting y-coordinate
    width = frame.shape[1] - 10  # Width of the cropped region
    height = frame.shape[0] - 42  # Height of the cropped region
    # Crop using slicing
    cropped_frame = frame[y_start:y_start + height, x_start:x_start + width]
    return cropped_frame

def resize_and_gray_cv2(frame, newsize, gray=False):
    frame = crop_Atari_frame_cv2(frame)
    if gray:
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    return cv2.resize(frame, newsize, interpolation=cv2.INTER_AREA)
```

2.2. Prioritized Experience Replay (PER)

- Added `PrioritizedReplayBuffer` class with:
- Priority-based sampling
- TD-error based priority updates
- Alpha parameter for controlling prioritization
- Memory sampling returns indices for priority updates

```
class PrioritizedReplayBuffer:
    def __init__(self, capacity, alpha=0.6):
        self.capacity = capacity
        self.buffer = []
        self.priorities = []
        self.alpha = alpha  # Controls how much prioritization matters (0 =
uniform, 1 = full prioritization)
```

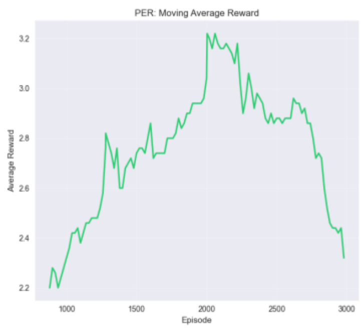
2.3. Frame Skipping to reduce computational load

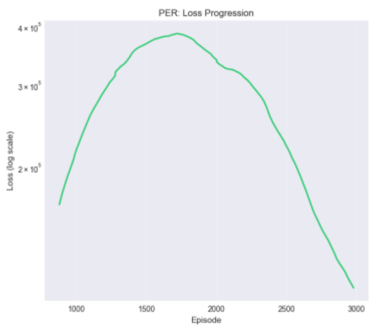
```
accumulated_reward = 0
for _ in range(4):  # Skip 4 frames (including the first one)
    next_state, reward, done, _, _ = env.step(action)
    accumulated_reward += reward
    if done:
        break
next_state = resize_and_gray_cv2(next_state, newsize, grayscale)
next_state = np.expand_dims(next_state, axis=0)
total_reward += accumulated_reward  # Add accumulated reward
```

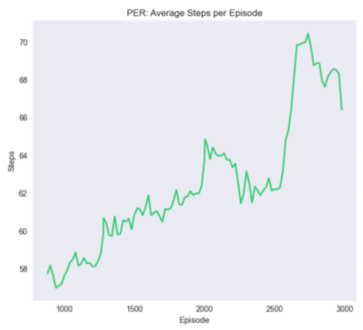
2.4. Training Improvements

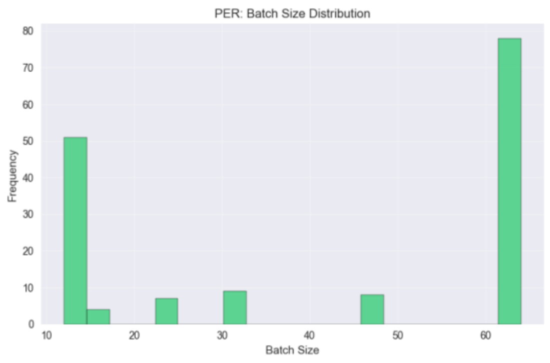
- Dynamic batch size (12-64) based on Loss
- More frequent target network updates (every 500 steps)

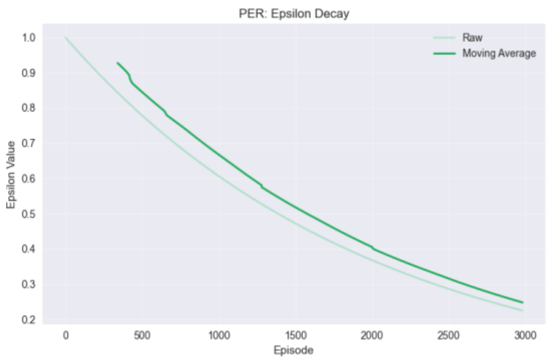
Parameter	Base	PER	Why
Memory Size	10,000	100,000	More experiences for better sampling
Learning Rate	0.001	0.0001	Stability in learning
Batch Size	Fixed 32	Dynamic 12-64	Adaptive to memory size
Target Update	1000	500	More frequent updates











- Added loss threshold monitoring
- Network update frequency of 32 steps
- Lower epsilon min (0.001 vs 0.01)
- Higher memory capacity (100,000)
- Batch Predication

2.5 Key Observations:

- Average Reward: The steady increase from the start until around episode 2000 peaking at 3.2 suggests that the agent is learning effectively, benefiting from PER. The decline could may be indicate exploration-exploitation balance issues.
- The rise in loss corresponds to the learning phase, where the agent is making significant updates to its Q-values in the episodes 1700-2000. The steady decline suggests the agent is converging towards an optimal policy.
- Steps per Episode: The gradual increase in steps per episode, peaking around episode 2700, suggests the agent is surviving longer and making better decisions. The late-stage dip may indicate overfitting or reduced exploration.
- Batch Size Distribution: The dominance of extreme batch sizes (12 and 64) reflects a balance between rapid learning with small batches and stable updates with larger batches. This suggests that PER is optimizing training dynamics but may benefit from more regularized batch selection.

3. OPT Version (580_hw5_OPT.ipynb) ran for approx 4 hrs on GPU

For this version, I removed the complex PER part and switched back to the original Memory Class and still kept a few optimizations like OpenCV usage, frame skipping, dynamic batch size, update frequency, gradient clipping, and batch prediction but with a minor tweaks.

Less running time could be because of GPU which is almost 2.5 to 3 times faster than CPU running in every experiment and also other factors contributing to faster processing would be removal of PER, updating frequency and batch size

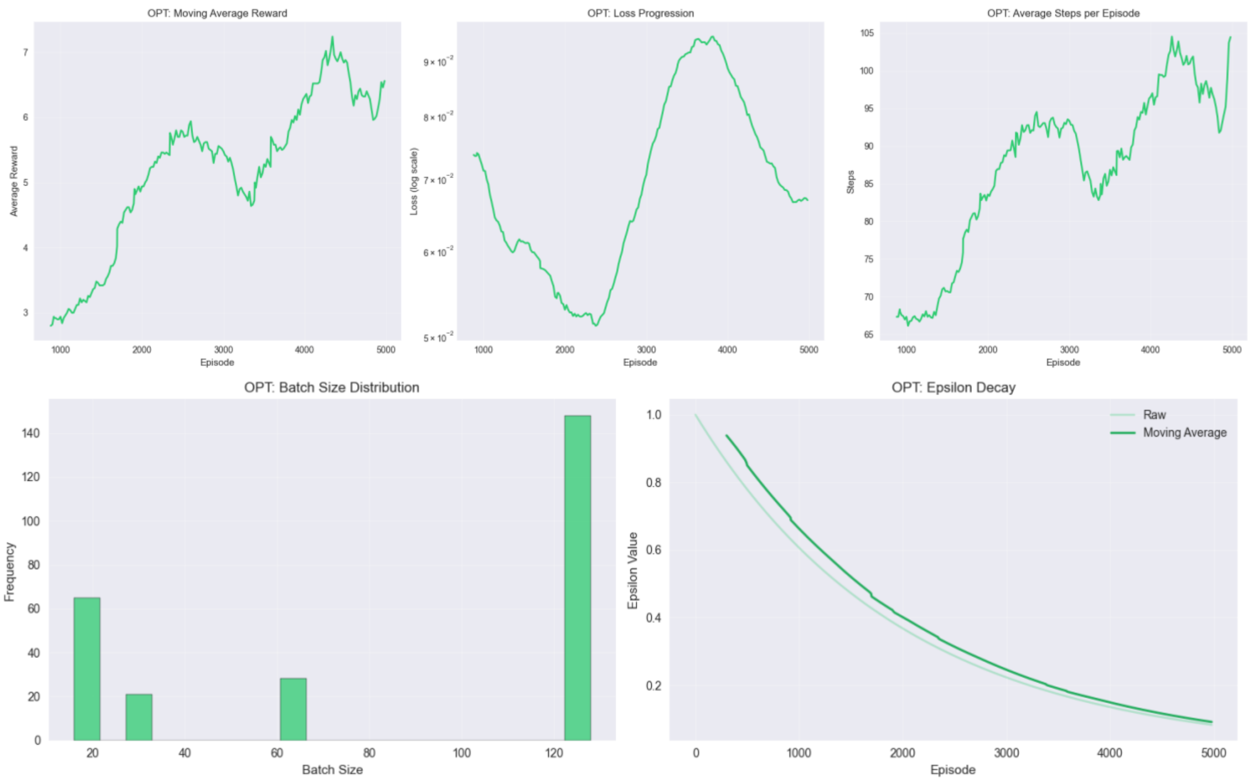
Further optimizations:

3.1. Memory & Processing

- Increased memory size (100,000)
- More aggressive dynamic batch sizing (16-128)

3.2. Training Parameters:

- Lower learning rate (0.0001)
- Less frequent target updates (1000 steps)
- More episodes (5000)



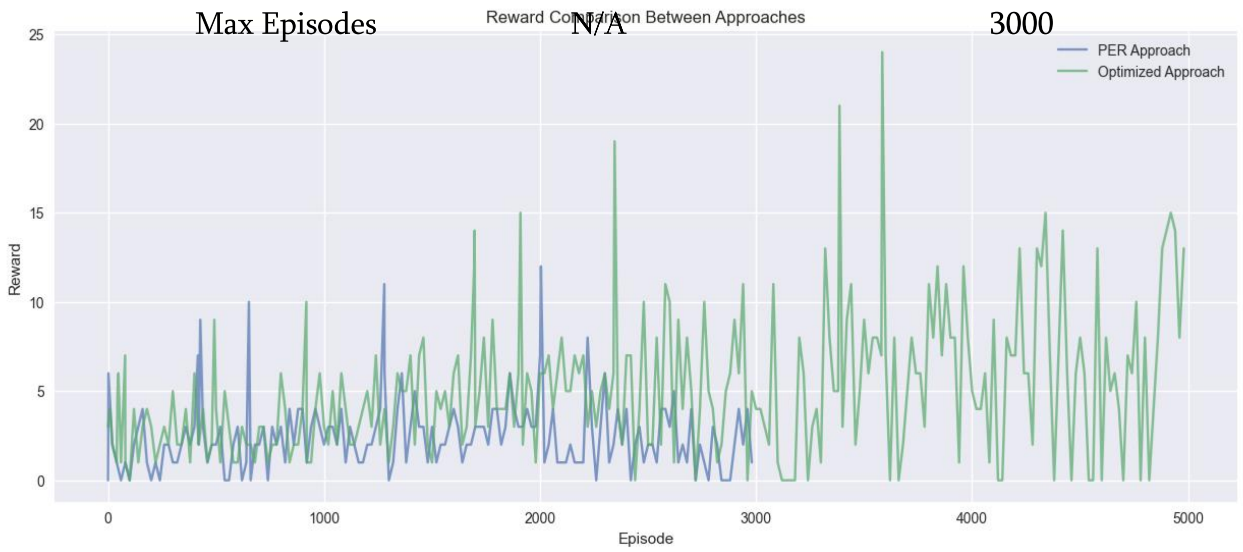
- More frequent network updates (every 4 steps)
- Higher epsilon min (0.01)

3.3. Key Observations:

- Average Reward: The steady increase in reward across episodes, peaking above 7, suggests improved learning efficiency due to optimizations. The occasional dips indicate potential instability but are followed by recovery, implying resilience in training. And actual rewards touched 20+ mark several times.

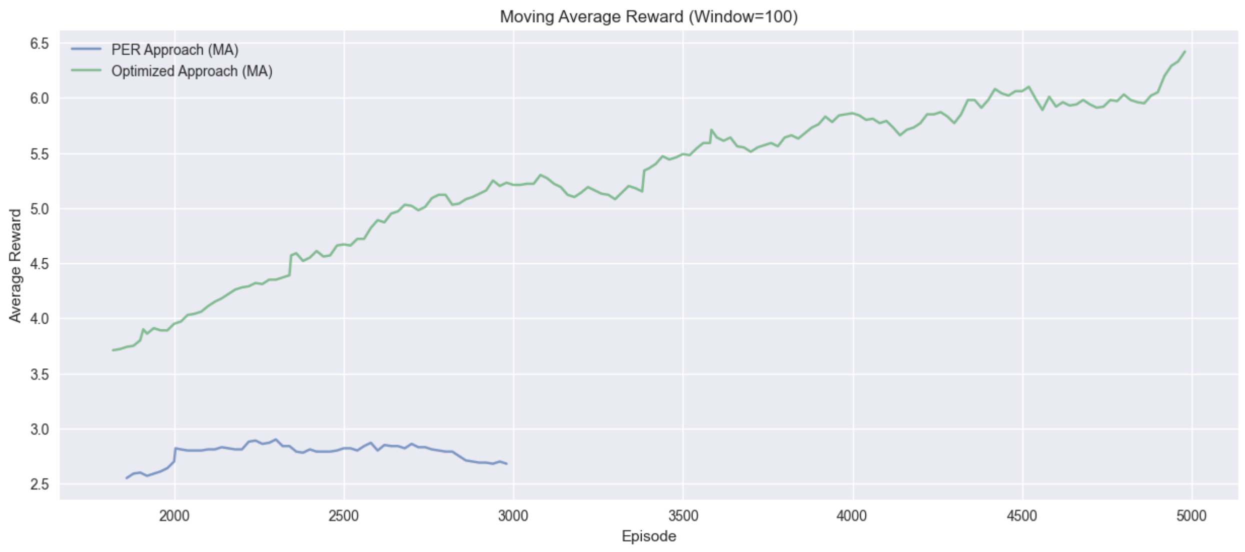
- Loss Progression: The fluctuating loss pattern, with a significant drop around episode 2000 and a rise before 4000, suggests periodic instability. However, the final decline indicates that the model stabilizes over extended training.
- Steps per Episode: The consistent rise in steps per episode, surpassing 100 steps, reflects stronger agent performance and prolonged survival in the environment. The minor fluctuations near the end could indicate slight instability in decision-making.
- Batch Size Distribution: The dominance of batch sizes at 16 and 128 suggests a preference for either rapid adaptation (small batches) or stabilized updates (large batches), reinforcing the dynamic batch sizing strategy.

Parameter	Base Model	PER Model	Optimized Model (OPT)
Memory Size	10,000	10,000	100,000
Batch Size Range	Fixed 16	12 - 64	16 - 128
Learning Rate	0.001	0.001	0.0001
Target Update	N/A	Every 500 steps	Every 1000 steps
Epsilon Min	N/A	0.001	0.01
Update Frequency	N/A	Every 32 steps	Every 4 steps
Max Episodes	N/A	3000	5000

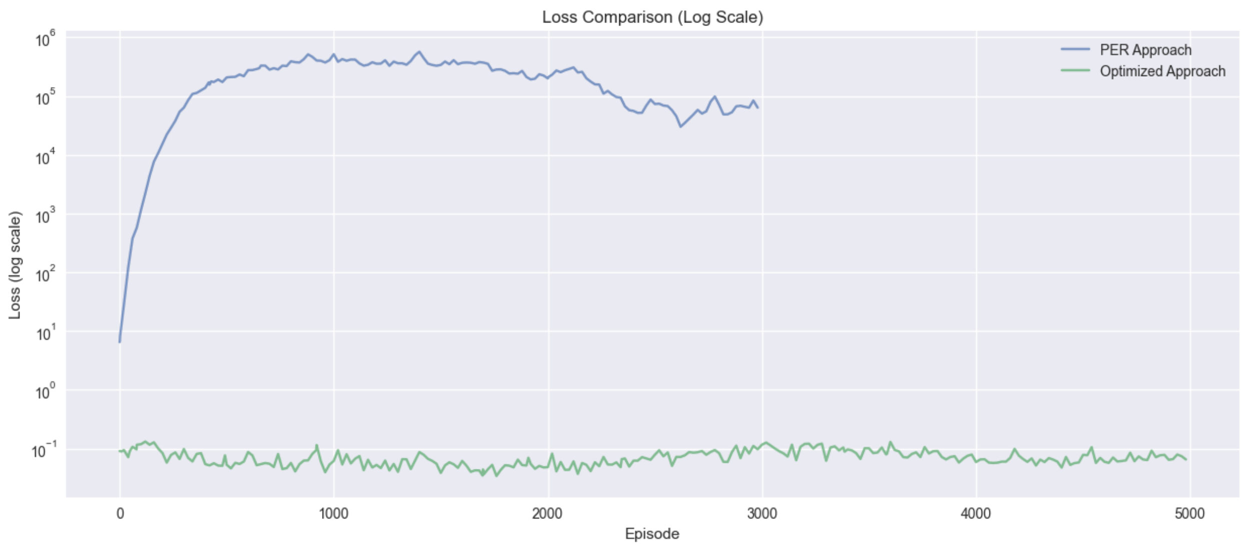


- Epsilon Decay: The smooth decay trend confirms a controlled shift from exploration to exploitation, with minimal fluctuations, suggesting well-tuned learning parameters.

Lets Compare both Outputs / Models

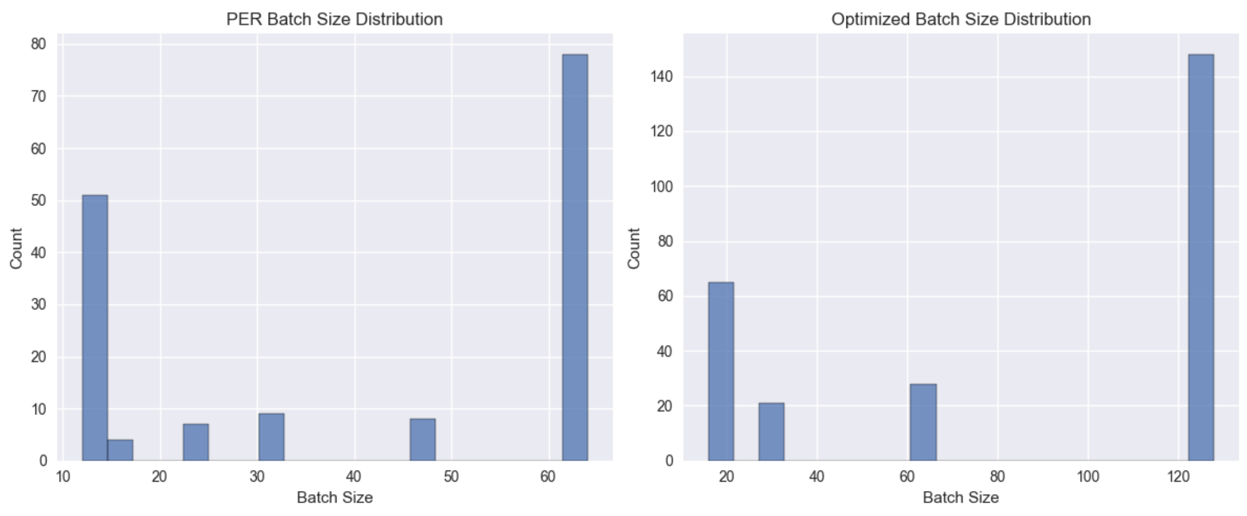


The Optimized Approach appears to explore more aggressively, leading to higher rewards at times, but with greater variance. The PER Approach ensures a more stable reward trend but does not reach the same high peaks.



The Optimized Approach outperforms PER significantly, achieving double the average reward by the end of training. This highlights the effectiveness of further optimizations in long-term learning stability and reward maximization.

The Optimized Approach significantly outperforms PER in terms of stability, avoiding excessive loss spikes and achieving smoother convergence. This highlights the effectiveness of lower learning rates, more frequent updates, and refined training parameters in preventing instability.



The Optimized Approach benefits from larger batch sizes, likely contributing to lower loss and better reward progression. Meanwhile, the PER Model's smaller batch sizes could explain its higher instability and slower learning curve.

The Optimized Model (OPT) outperforms the PER Model in reward progression, training stability, and computational efficiency. This success is attributed to several key improvements:

Memory and Batch Size Adjustments

- Larger batch sizes (16-128 vs. 12-64 in PER) helped in stabilizing training by reducing variance in weight updates.

Refined Training Dynamics

- Higher update frequency (every 4 steps vs. 32 in PER) allowed the network to adapt faster without excessive stale learning.
- Less frequent target network updates (every 1000 steps vs. 500 in PER) prevented overfitting to short-term experiences, leading to better policy convergence.

Improved Learning Stability

- The lower learning rate (0.0001 in OPT vs. 0.001 in PER) resulted in smoother weight updates, preventing large fluctuations in loss and ensuring better long-term stability.
- Gradient clipping and loss monitoring further prevented instability, which was more prevalent in PER due to its prioritization mechanism.

And I think model would perform really well if we allow it to run more episodes

Next steps?

1. Exploring different architectures
2. Longer training
3. Dynamic Epsilon

What I learned

1. DQN in practice
2. PER vs Standard Experience Relay
3. Balancing Epsilon
4. Computational Efficiency Importance
5. And more..

This assignment reinforced the importance of structured experimentation, efficient training strategies, and continuous hyper parameter tuning in reinforcement learning. The hands-on approach provided valuable insights into optimizing DQN models while balancing performance and computational efficiency.