Reinforcement Learning Programming - CSCN8020 - Assignment 3

Dheeraj Choudhary

ID: 9014533

Introduction

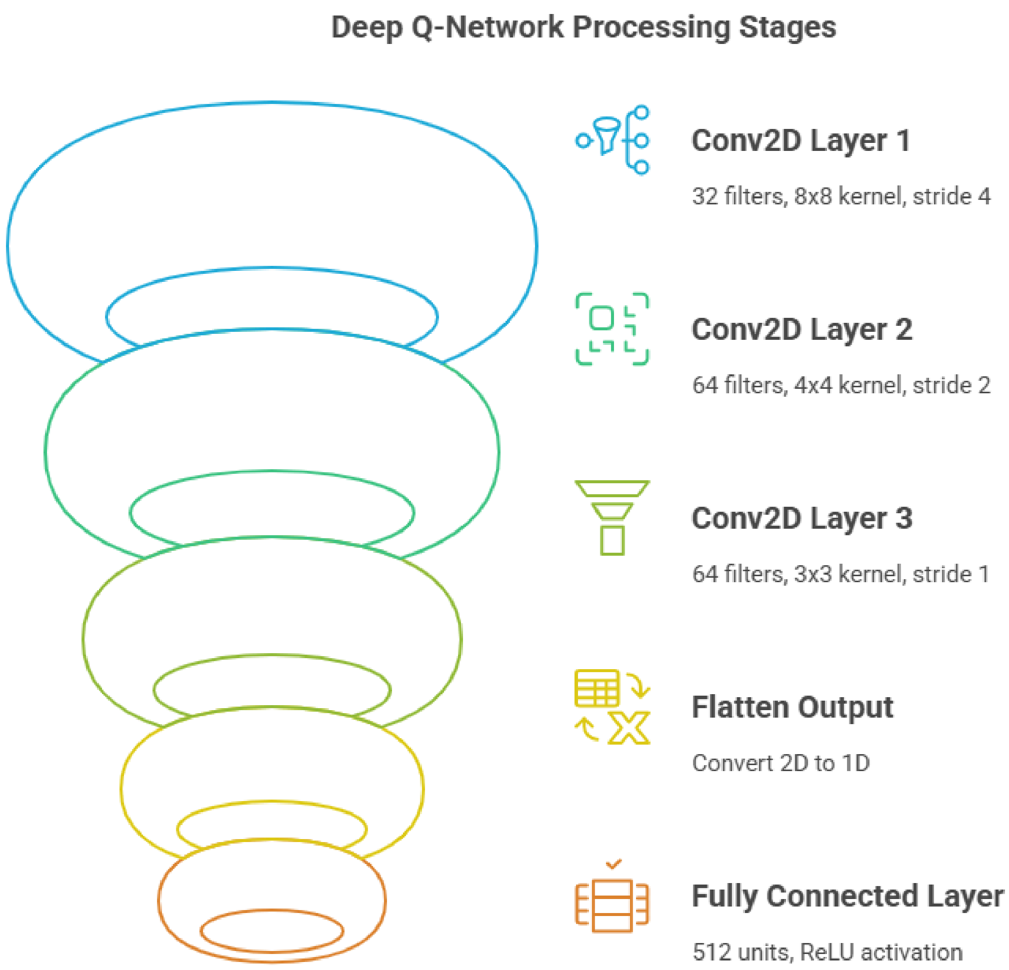
In this assignment, we implement **Deep Q-Learning (DQN)** to train an AI agent to play **Pong** using **OpenAI Gym's PongDeterministic-v4 environment**. Since Pong has a continuous state space, traditional Q-learning is not feasible, so we use a **Deep Neural Network (DNN)** to approximate the Q-values.

The key objectives of this assignment are:

- **Preprocess game frames** by cropping, converting to grayscale, and normalizing them.
- **Implement a Deep Q-Network (DQN)** with convolutional layers to process image-based inputs.
- **Train the agent** using experience replay and target network updates.
- **Experiment with hyperparameters** such as batch size and target network update frequency.
- **Evaluate the agent's performance** by tracking the reward progression over episodes.

Network Architecture

The Deep Q-Network (DQN) implemented in this project processes a stack of four grayscale image frames as input. It uses three convolutional layers to extract spatial features, followed by a flattening operation and two fully connected layers to produce Q-values for each possible action. ReLU activation is applied after each layer except the output. The architecture is designed to efficiently learn value estimates from raw pixel data in a reinforcement learning environment.



Layer Type	Filters/Units	Kernel Size	Stride	Activation
Conv2D	32	8×8	4	ReLU
Conv2D	64	4×4	2	ReLU
Conv2D	64	3×3	1	ReLU
Flatten	—	—	—	—
Fully Connected	512	—	—	ReLU
Output Layer	num_actions	—	—	None

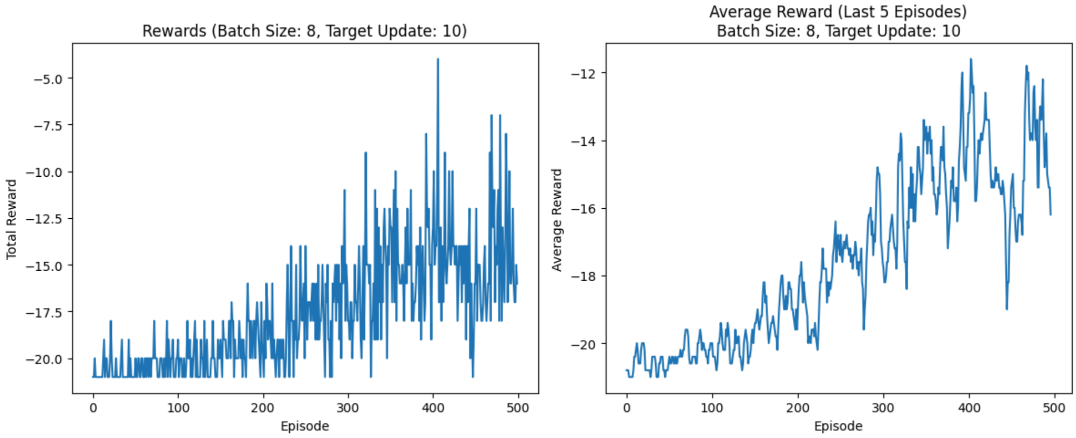
- **Input Shape:** `(4, 84, 80)` — A stack of 4 preprocessed grayscale frames, each of size 84×80 pixels.
- **Output:** A vector of Q-values representing possible actions (`num_actions`).
- **Activation Functions:** ReLU is used after each convolutional and the first fully connected layer.
- **Flatten Layer:** Applied before passing to fully connected layers to convert 3D feature maps into 1D.

📊 Training Metrics Visualizations

Below are the reward trends and average reward comparisons for all four experiments:

📈 Experiment 1: Default (Batch Size = 8, Target Update = 10)

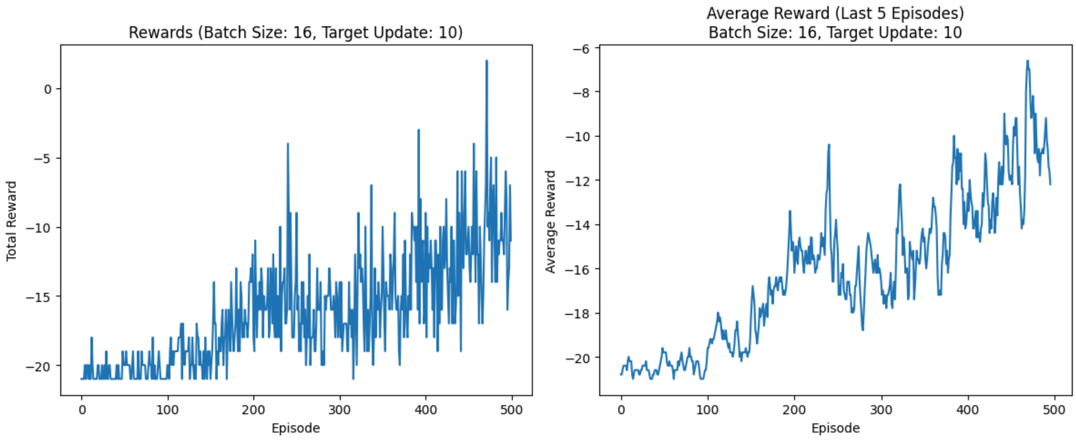
Total Rewards per Episode and Average Reward (Last 5 Episodes)



- The initial rewards were very low (-21), showing poor performance.
- Over time, the agent showed gradual improvement, reaching an average reward of **-12.2** at episode 490.
- The improvement was slow but steady, indicating stable but suboptimal training.

📈 Experiment 2: Batch Size 16, Target Update = 10

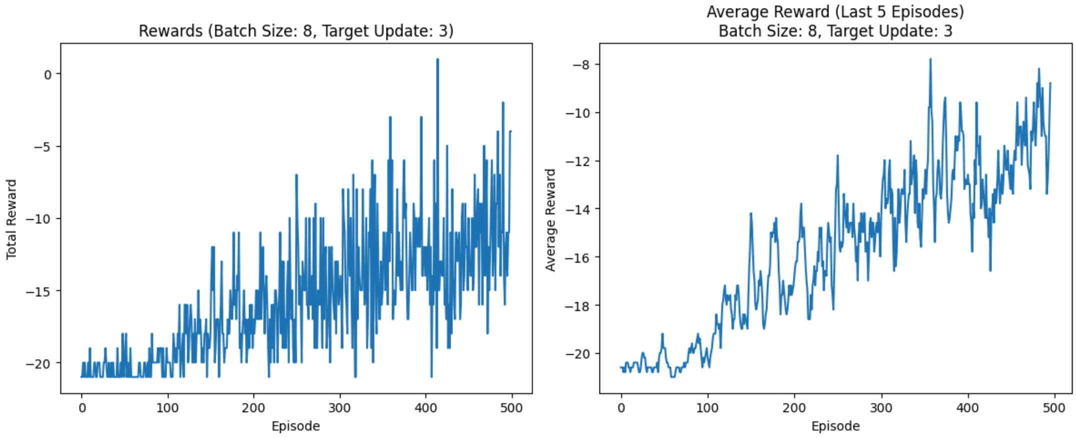
Total Rewards per Episode and Average Reward (Last 5 Episodes)



- Initially, the agent performed similarly to the default settings.
- By episode 490, the average reward reached -10.6, which is slightly worse than the default setting.
- Larger batch sizes may have caused slower updates, leading to suboptimal performance.

📈 Experiment 3: Batch Size 8, Target Update = 3

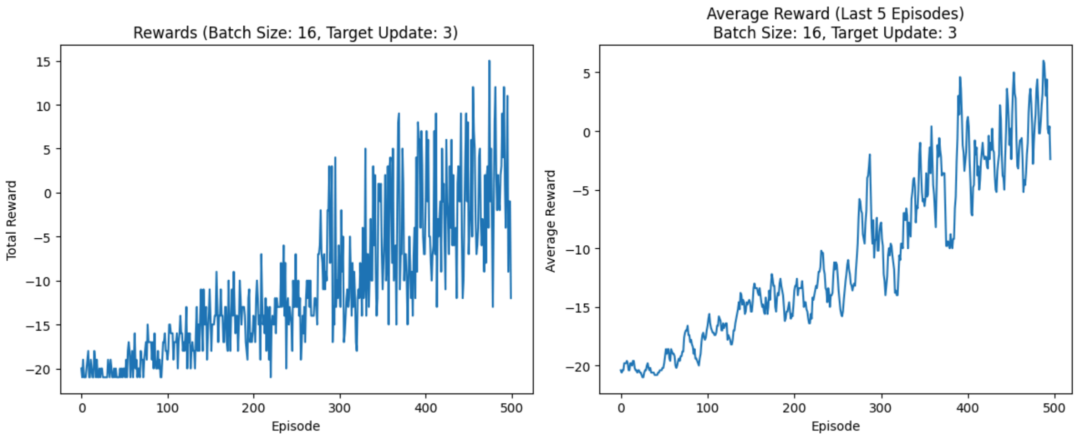
Total Rewards per Episode and Average Reward (Last 5 Episodes)



- The agent started similarly to the other experiments.
- Training was significantly more effective, with the average reward improving to -9.0 by episode 490.
- More frequent updates to the target network accelerated learning.

Experiment 4: Best Combo (Batch Size = 16, Target Update = 3)

Total Rewards per Episode and Average Reward (Last 5 Episodes)



- The model showed slow learning initially but rapidly improved after episode 280.
- It reached a peak average reward of **3.2**, the highest among all experiments.
- This confirms that the combination of a larger batch size with frequent target updates yields the most effective learning.

Detailed Analysis of Observations

1. Impact of Batch Size on Learning Speed

Increasing the batch size from 8 to 16 initially showed slower learning, but when combined with frequent target updates, it led to strong performance improvements in later episodes.

2. Effect of Target Network Update Frequency

Frequent target updates (every 3 episodes) improved learning stability and responsiveness. This effect became more pronounced when paired with a larger batch size.

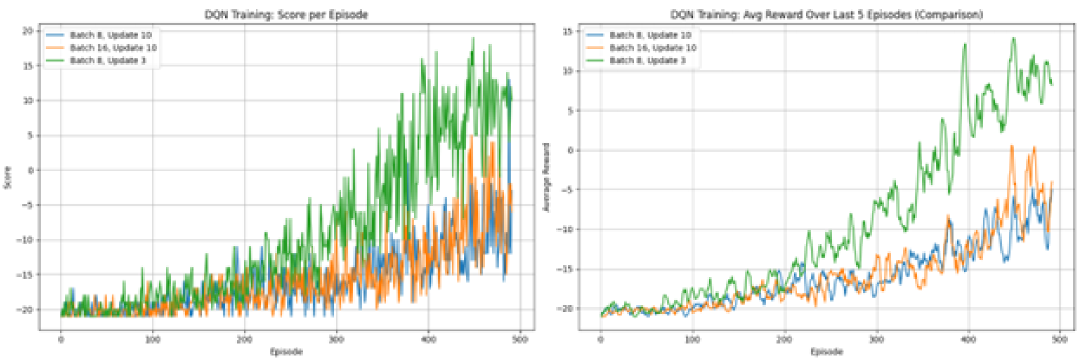
3. Convergence Behavior

Experiment 4 (Best Combo) showed a sharp improvement after episode 280, reaching a final **average reward of 3.2**, outperforming all prior settings. This indicates that the model adapted better with larger batch sizes when updated more frequently.

4. Comparison of Experiments

Experiment	Batch Size	Target Update	Avg Reward (Last 5)
Default	8	10	-12.2
Batch 16	16	10	-10.6
Target 3	8	3	-9.0
Best Combo	**16**	**3**	**3.2 (Best)**

From the table, the best results were achieved using **batch size = 16** and **target update = 3**.

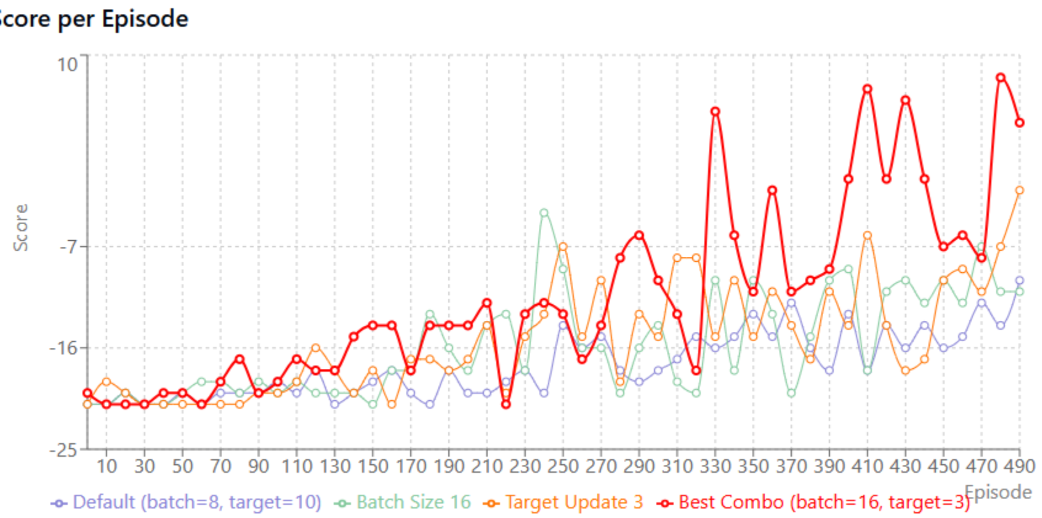


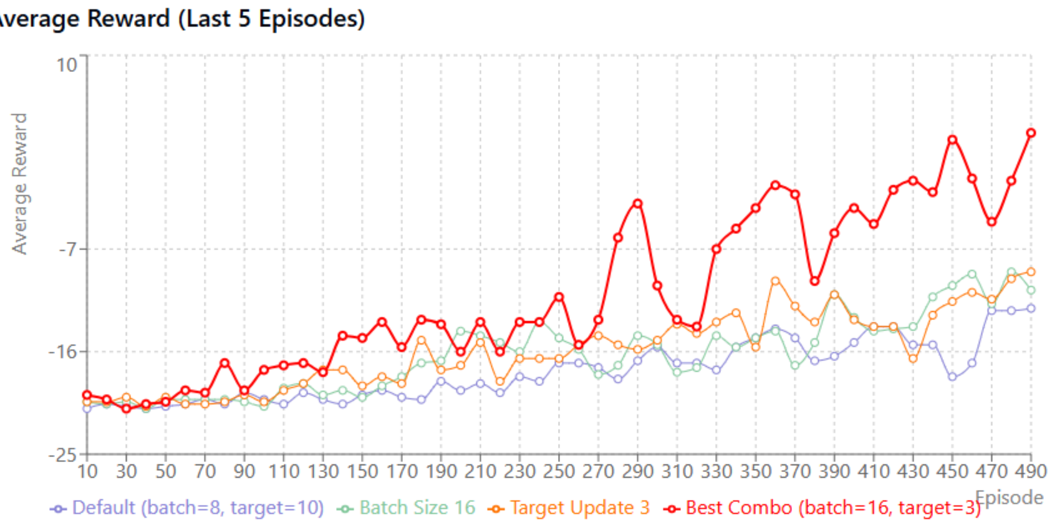
5. Justification for Best Configuration

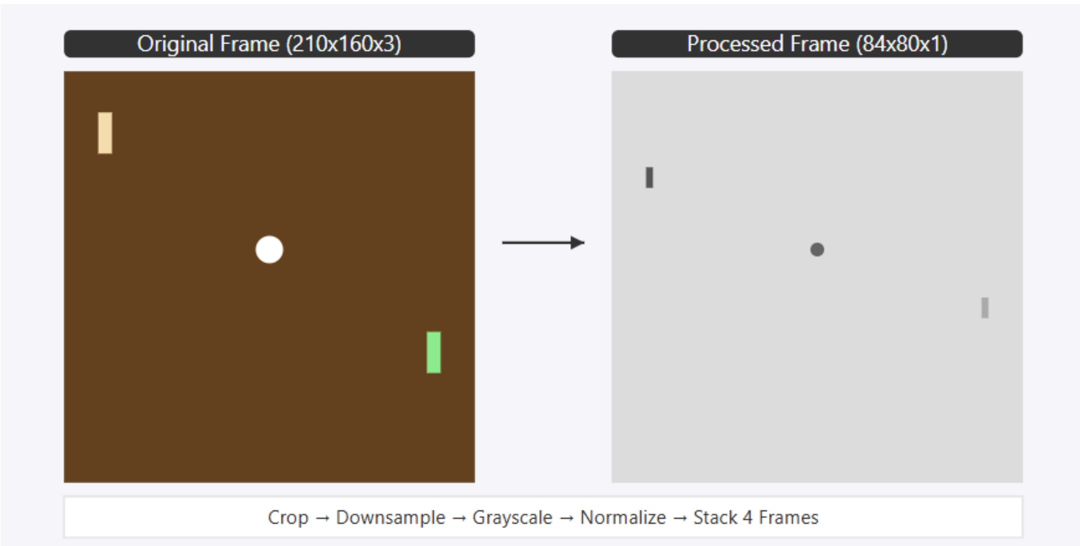
The best-performing setup was **batch size = 16, target update = 3** due to:

- Stable early learning.
- Rapid improvement post-episode 280.
- Achieving positive rewards by the end of training.

Deep Q-Network Performance Comparison







Conclusion

The experiment confirms that **larger batch size combined with frequent target updates** leads to optimal performance in this DQN setup.

The **best configuration for this task** is:
✅ **Batch Size = 16, Target Update = 3**

This combination led to the highest final performance with an **Avg Reward (Last 5) of 3.2**, outperforming other settings.