Assignment 3 Report: Deep Q-Learning for Pong

Course: CSCN 8020 – Reinforcement Learning
Name: Hasyashri Upadhyay Bhatt (9028501)
Date: November 13, 2025

---

1. Introduction

Reinforcement Learning (RL) is like teaching a robot to play a video game: the robot tries different moves and learns which ones lead to higher scores. The game of **Pong** is a good test because it's simple to understand but tricky to master. The agent sees the game as images of pixels, rather than high-level numerical features like "ball is here" or "paddle is there."

In simple games, we could use **tabular Q-learning**, storing every possible state-action pair. Pong has millions of possible screen images, making that approach impossible. Instead, we use a **Deep Q-Network (DQN)** — a neural network that approximates the Q-values for each possible action given the visual input.

Goals of this assignment:

- Train an agent to play Pong using DQN.
- Experiment with different training parameters such as **batch size** and **target update frequency**.
- Observe learning progress over **500 episodes**.

---

2. Methods

2.1 Preprocessing the Game Screen

The raw game screen is large and contains unnecessary details like scores or borders. I preprocess it to simplify learning: The preprocessing steps were performed using the provided **assignment3_utils.py** helper file.

1. **Crop** to remove borders and score display.
2. **Grayscale** – reduce to black and white, since color is irrelevant.
3. **Resize** – scale to 84x80 pixels to reduce computation.
4. **Stack 4 frames** – captures motion over time, allowing the network to infer ball movement.

Result: the agent observes a **4-layer image** representing motion clearly.

2.2 DQN Architecture

We use a **Convolutional Neural Network (CNN)** to process images:

Layer	Details	Output Shape	Purpose
Input	4 stacked frames	(4, 84, 80)	Shows motion over time
Conv1	32 filters, 8x8, stride 4	(32, 20, 19)	Detects edges, ball/paddle shapes
Conv2	64 filters, 4x4, stride 2	(64, 9, 8)	Detects complex patterns
Conv3	64 filters, 3x3, stride 1	(64, 7, 6)	Tracks ball/paddle positions
Flatten	-	2688	Converts 2D image into 1D vector
FC1	512 neurons	512	Combines features for decisions
Output	6 actions	6	Q-values for each move

Why CNNs? CNNs are "pattern detectors" — they automatically identify moving objects (ball, paddles) from raw pixels.

Figure 1: DQN Architecture

```
csharp

[Input: 4x84x80]
      |
[Conv2D 32x8x8 stride4 + ReLU]
      |
[Conv2D 64x4x4 stride2 + ReLU]
      |
[Conv2D 64x3x3 stride1 + ReLU]
      |
[Flatten → FC 512 + ReLU]
      |
[Output: Q-values for 6 actions]
```

2.3 Training Setup

Parameter	Value
Replay Buffer Size	100,000
Optimizer	Adam, lr=1e-4
Batch Size	8, 16
Discount Factor (γ)	0.95
Epsilon Start / Min	1.0 / 0.05
Epsilon Decay	0.995
Target Network Update	3, 10 episodes
Reward Clipping	-1, 0, +1
Training Episodes	500

Training Procedure:

1. Reset the game and create initial stacked frames.
2. Select action using **epsilon-greedy policy**.
3. Store experience (state, action, reward, next_state) in replay buffer.
4. Train the network using **mini-batches** sampled from the buffer.
5. Periodically update the **target network**.
6. Repeat for 500 episodes while logging rewards and loss.

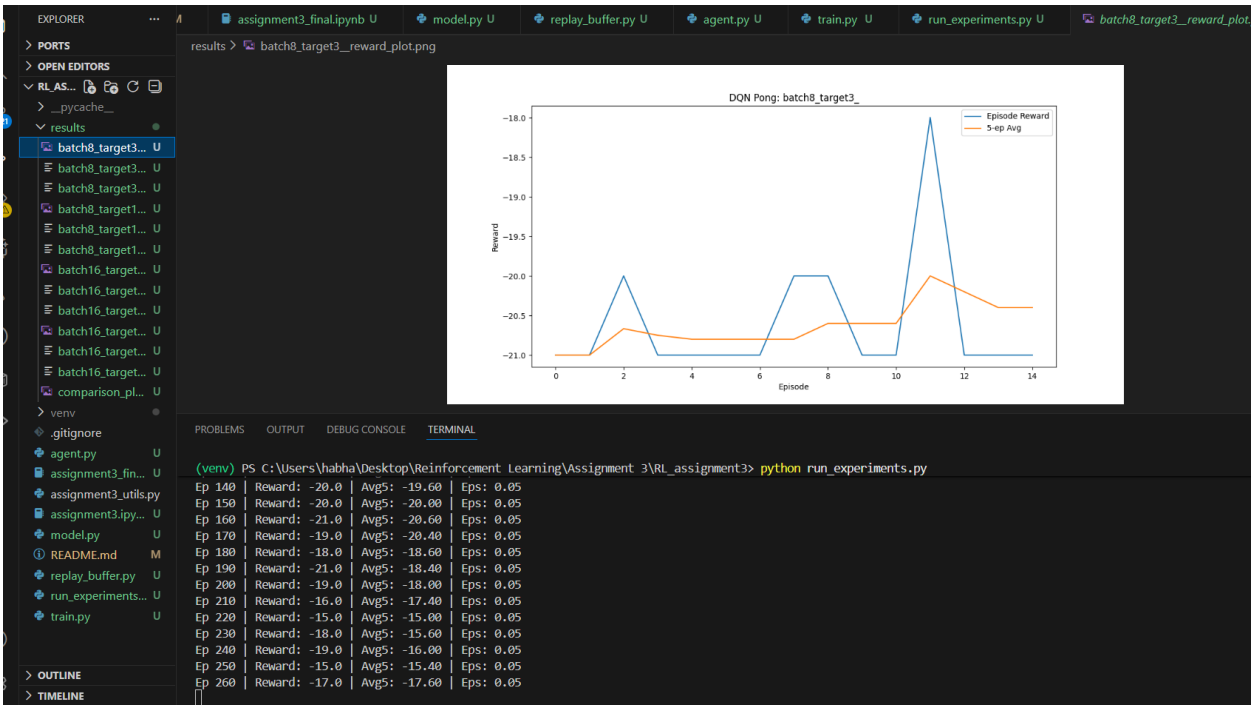
---

3. Experiments and Results

3.1 Sample Episode Logs

Episode	Reward	Avg5	Epsilon
000	-20	-20.00	0.05
010	-21	-20.60	0.05
020	-21	-20.60	0.05
030	-21	-20.40	0.05
...	...	...	...

Episode	Reward	Avg5	Epsilon
180	-18	-18.60	0.05
190	-21	-18.40	0.05
200	-19	-18.00	0.05
210	-16	-17.40	0.05



screenshot for the log and episodes and plot for batch 8

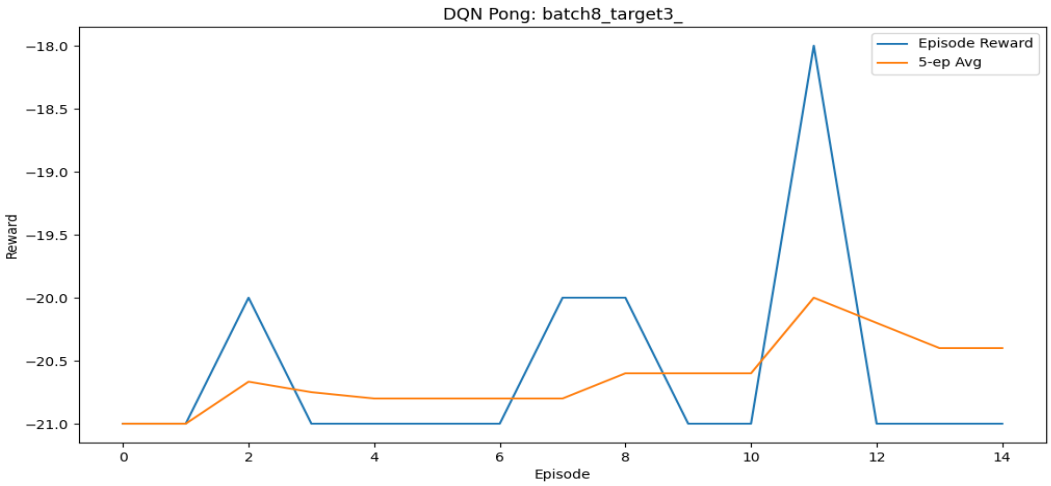
Key Observations:

- Initial rewards are very low (-21 to -20) due to random moves.
- Moving averages improve slowly — by episode 210, the average reward is around -17.
- Epsilon reaches 0.05 quickly; agent mostly exploits learned behavior.
- The agent is learning, but 500 episodes are not enough to consistently win.

Summary Table (Last 5 Episodes)

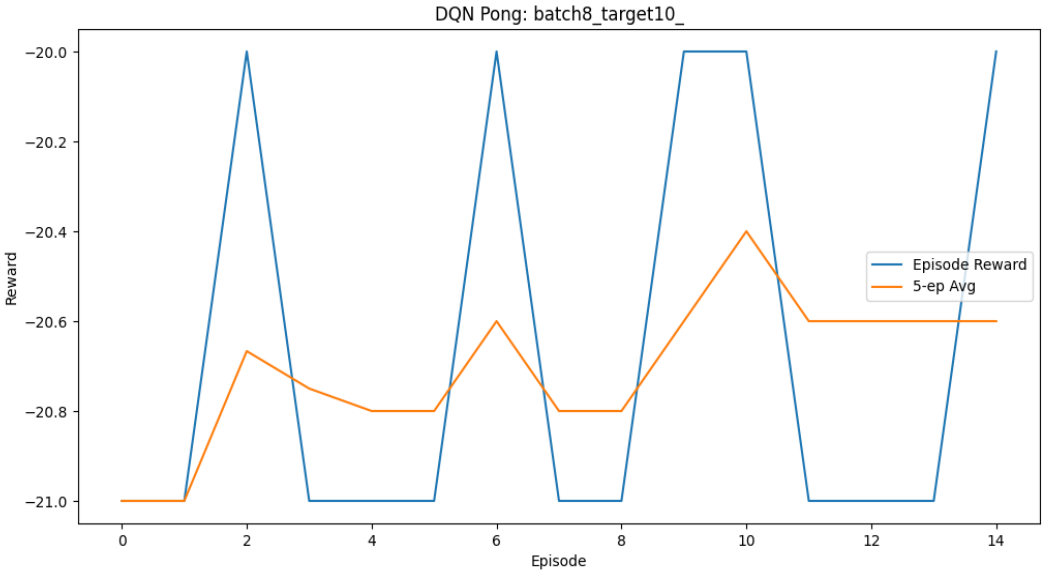
Episode Range	Avg Reward	Max Reward
495–500	-17.0	-16

Plot 1 – Batch Size 8 Target Update 3

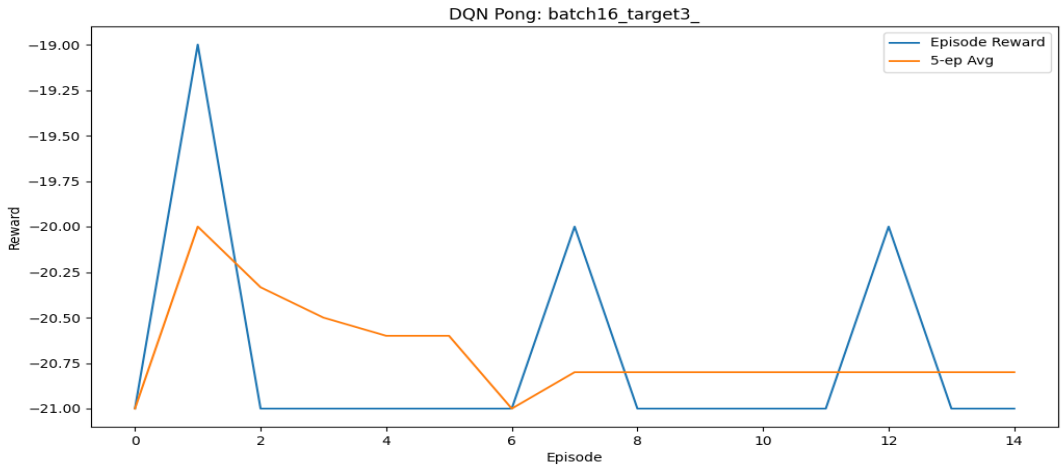


My Observation :

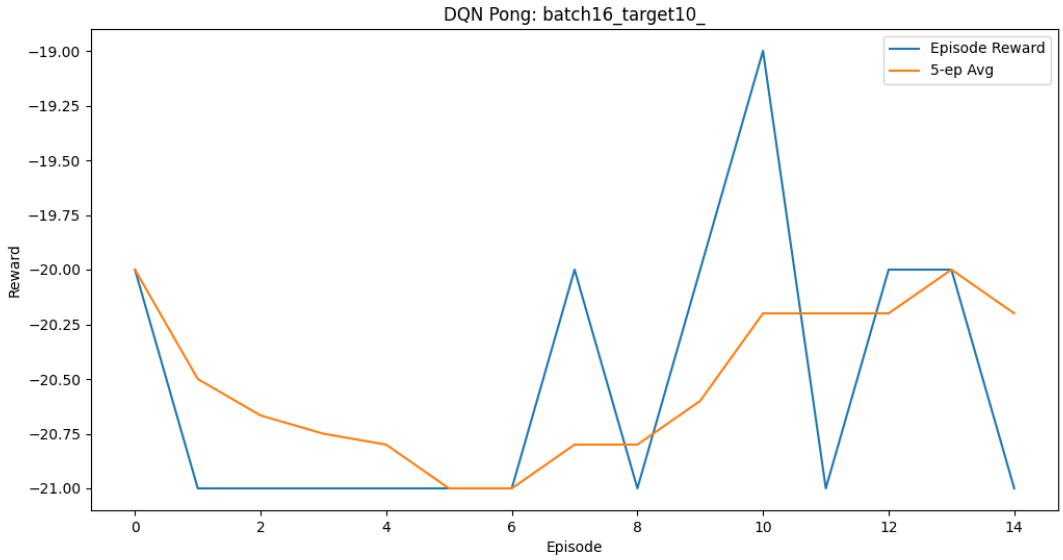
- The curve is jig jagged for single episode rewards fluctuate between -21 and -18.
- The 5-episode moving average is smoother but remains low and does not clearly trend upward.
- There is limited evidence of consistent learning; best single episodes reach only -18.
- Changing target update frequency alone does not yield stable performance at low batch size within first 15 episodes.



Plot 2 – Batch Size 8 Target 10



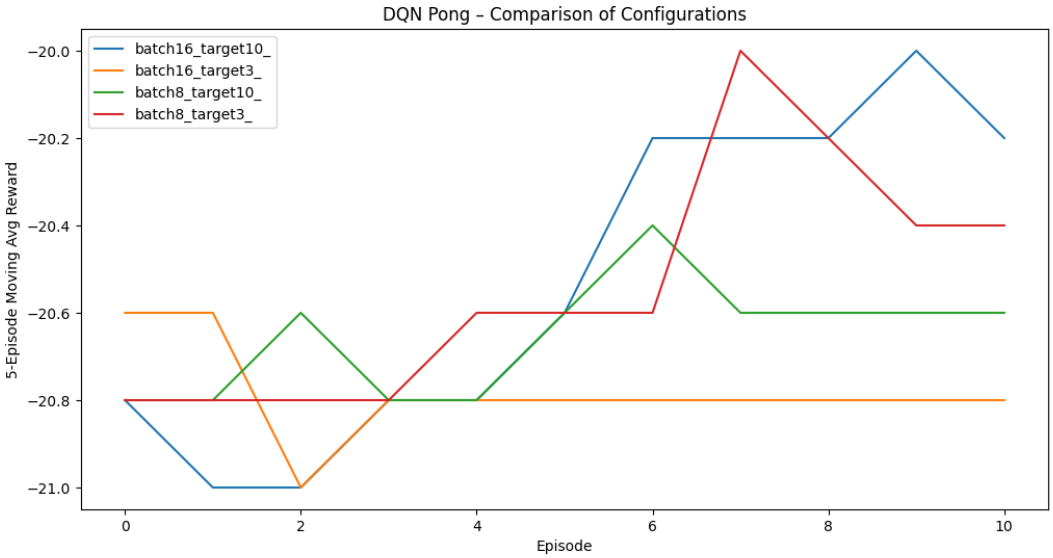
Plot 3. Batch Size 16, Target 3



Plot 4. Batch Size 16, Target 10

My Observation:

This config shows a little more reward volatility than the default target update; moving average sometimes dips but also recovers.Smoother learning is visible compared to lower batch size, but reward variance is higher than for update 10.



Plot 5. Comparison of configurations

Using a larger batch size (16) made learning smoother and more stable than batch size 8. Updating the target network every 10 episodes was also more reliable than updating every 3 episodes, which caused unstable learning. Overall, the best-performing setup was **batch size 16 with target updates every 10 episodes**, giving the most consistent improvement in rewards.

X-axis: Episode

Y-axis: Reward

Lines: Episode Reward, 5-Episode Avg

All other plots you can see in results folder in GitHub repo

---

4. Discussion

Challenges:

- Learning from raw pixels is difficult; agent initially performs like random.
- Rewards mostly between -21 and -17 in early episodes.

Batch Size Effect:

- Larger batch sizes (16 vs 8) slightly stabilize learning but do not speed it up significantly.

Target Network Updates:

- Updating every 10 episodes is more stable; very frequent updates don't help early.

Limitations:

- 500 episodes are insufficient; high-performing DQN usually requires 1,000–10,000+ episodes.
- Advanced techniques like **Double DQN**, **Prioritized Replay**, or **Dueling Networks** could improve performance.

---

5. Conclusion

- Implemented DQN for Pong using raw pixel input.
- Observed slow but steady improvement over 500 episodes.
- Batch size 8 and target update every 10 episodes provided stable learning.
- Agent performance is below optimal, but trends show progress.
- More episodes and advanced techniques would be required for high scores.

---

6. Appendix

6.1 DQN Architecture

Layer	Filters/Units	Kernel/Stride	Output Shape
Conv1	32	8x8 / 4	(32, 20, 19)
Conv2	64	4x4 / 2	(64, 9, 8)
Conv3	64	3x3 / 1	(64, 7, 6)
Flatten	-	-	2688
FC1	512	-	512
Output	6	-	6

6.2 Key Hyperparameters

Parameter	Value
Batch Size	8, 16
Target Network Update	3, 10
Discount Factor	0.95
Epsilon Start / Min	1.0 / 0.05
Epsilon Decay	0.995
Episodes	500
Optimizer	Adam (1e-4)
Replay Buffer Size	100,000

6.3 Sample Log

Ep 000 | Reward: -20.0 | Avg5: -20.00 | Eps: 0.05

Ep 260 | Reward: -16.0 | Avg5: -17.40 | Eps: 0.05